Die approbierte Originalversion dieser Dissertation ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

DISSERTATION

Queueing models for multi-service networks

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Em.O.Univ.Prof. Dipl.-Ing. Dr.techn. Harmen R. van As
e389 – institute of telecommunications
Technische Universität Wien

und

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Karl Grill
e107 – Institut für Statistik und Wahrscheinlichkeitstheorie
Technische Universität Wien

eingereicht an der

Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

Dipl.-Ing. Gerald Franzl
Matr.-Nr. 8526885

Wien, im Februar 2015

Abstract

The thesis starts in chapter 1 with basic observations and a survey of statistical methods commonly used to express the performance of communication network components and other systems based on stochastic processes. Multi Protocol Label Switching (MPLS) is introduced and further on used as example toward a modern multi-flow based network performance assessment.

In chapter 2 we first introduce some basic distributions to model the inter-arrival and holding time distributions, and second, we present methods to construct or approximate more complex processes by mathematically tractable Markov chains.

Based thereon we then exemplify basic queueing models in chapter 3, as they can be found in most text-books on queueing theory, though hardly in a similar synopsis. Starting with infinite multi-server systems we continue to the more practical finite and queue-less systems, and learn about different approaches to analyse these systems based on the Markov chains that result from the state transition diagrams representing these systems in a comprehensible manner, besides other methods applied where Markov chains are not an option.

Finally, in chapter 4 we use the introduced concepts and methods to evaluate the principal mechanisms of multi-service communication network components. Form resource sharing and flow prioritisation we proceed to congestion mitigation and finally chains and networks of queueing systems.

In the end, in section 4.4, we outline a subjective vision of future network management concepts based on the observation that the main cause for service degradation is overload, and the intuitive conclusion that this obstacle cannot be mitigated without proper network management. The basic instantaneous connectivity demand problem is assumed to be solved by connection advertisement based on pre-configured intra section paths and pre-calculated expectable service quality for the advertised end-to-end paths.

The load balancing issue is not solved in detail because a reasonable discussion of the routing topic, required to achieve load balancing, would add at least one more lengthy chapter. Instead, based on previous work, it is assumed that the introduction of a common virtual network level in the management hierarchy solves the scalability problem and that per autonomous network section capable algorithms alike the *Residual Network and Link Capacity* algorithm can be applied with reasonable computation effort.

Chapter 5 concludes the thesis, reflecting on the course of preparing the thesis and the findings and obstacles identified. The addenda provide examples of the *Octave* programs used to calculate the many figures on system performance metrics, in particular the event based system simulation routine and an example on the matrix oriented design of multidimensional state transition diagrams.

Zusammenfassung

Die Doktorarbeit beginnt in Kapitel 1 mit grundlegenden Beobachtungen und einer Zusammenfassung der statistischen Methoden die üblicherweise zur Bestimmung der Leistungsfähigkeit eines Kommunikationsnetzes Verwendung finden. Vielfachprotokollmarkierungsumschaltung (MPLS) wird vorgestellt und im weiteren als Beispiel für die Effizienzbewertung von Netzkomponenten unter Berücksichtigung mehrerer Datenflüsse verwendet.

In Kapitel 2 werden erst grundlegende Wahrscheinlichkeitsverteilungsfunktionen vorgestellt welche zur Modellierung der Zwischenankunftszeiten und Bedienzeiten Verwendung finden. Im weiteren werden Methoden zur Modellierung komplexerer Prozesse durch die Verknüpfung einzelner negaitv exponentieller Phasen zu mathematisch handhabbaren Markovketten gezeigt.

Darauf basierend werden in Kapitel 3 grundlegende Warteschlangensysteme vorgestellt, wie sie auch in den meisten Lehrbüchern zu finden sind, wohl aber kaum in einer identischen Zusammenstellung. Beginnend mit unendlichen Warteschlangensystemen mit mehreren Verarbeitungsinstanzen werden im weiteren praxisgerechte endliche und warteschlangenlose Systems behandelt. In diesem Zusammenhang lernen wir verschiedene Möglichkeiten kennen wie derartige Systeme mithilfe von Markovketten analysiert werden können, also auch andere Methoden für Systeme die sich mittels Markovketten nicht modellieren lassen.

Schließlich, in Kapitel 4, werden die vorgestellten Konzepte und Methoden dazu verwendet die primären Mechanismen von Kommunikationsnetzknoten mit Dienstklassendifferenzierung zu untersuchen. Angefangen bei der gemeinsamen Nutzung der zur Verfügung stehenden Transportkapazitäten und der Flusspriorisierung kommen wir im weiteren zur Datenstauabarbeitung und schließlich zu verketteten und vernetzten Warteschlangensystemen.

Zu guter letzt, in Kapitel 4.4, wird eine subjektive Vision zukünftiger Netzverwaltungskonzepte umrissen, basierend auf der Beobachtung, dass Überlast der Hauptgrund für eine Dienstverschlechterung darstellt sowie der intuitiven Erkenntnis, dass dieses Hindernis nicht ohne eine gute Netzverwaltung ausgemerzt werden kann. Es wird davon ausgegangen, dass das grundlegende Problem der zeitnahen Bereitstellung von Datenverbindungen mittels vor-konfektionierter Verbindungsstücke und der Anpreisung der damit bereitstellbaren Ende-zu-Ende Dienstqualitäten gelöst werden kann.

Das Lastausbalanzierungsproblem wird hier nicht bis ins Detail gelöst da eine seriöse Darstellung des Wegefindungsproblemes, wie es nötig ist um Lastgleichverteilung zu erreichen, zumindest ein weiteres umfangreiches Kapitel bedürfen würde. Basierend auf früheren Arbeiten wird davon ausgegangen, dass die Einführung einer virtuellen Netzebene das Skalierungsproblem löst, und dass die Verwendung fähiger Algorithmen, wie zum Beispiel dem *Verbleibende Netz- und Link-Kapazität* basierten Algorithmus (*RNLC*), innerhalb der begrenzten Netzbereiche mit vertretbarem Rechenaufwand möglich ist.

Kapitel 5 beschließt die Dissertation und wirft einen Blick zurück auf die Vorbereitung sowie die identifizierten Erkenntnisse und Hindernisse. In den Ergänzungen finden sich Beispiele der *Octave* Programme die im Verlauf der Arbeit entstanden und Verwendung fanden zur Berechnung der vielen Systemqualitätskurven. Insbesondere sind das die ereignisgesteuerte Simulation und ein Beispiel für den Matrizen basierten Entwurf von mehrdimensionalen Zustandsübergangsdiagrammen.

Preface

Looking for a challenge, as I usually tend to do when I should not, I choose for my Ph.D. thesis a topic quite new to me. Having finishing my master thesis on *on-demand wavelength routing*, I found it coherent to switch to flow transmission performance. The courses given by Prof. van As presented the queueing systems topic well and it was clear that the approach is a key issue. A first attempt to extend the precise theory of tandem systems to chained systems I gave up the moment I realised that this is based on assuming independent service times at subsequent systems, which is never the case with chained communication channels. Queueing networks are also not very appealing because these demand random routing in addition to independent service times at subsequent nodes, which both contradicts with communication systems practice.

After leaving the university for a job in the private industry for several years, still continuing scientific work in the course of ongoing research projects, I returned and delved into the topic more broadly, backing off to a hop-by-hop, feature-by-feature approach. Initially, network calculus appeared utile to combine the results. However, the deterministic version that provides upper and lower performance bounds appears stigmatised as too restrictive. The true issue is that the mean performance cannot be calculated using this method. The stochastic network calculus promised to substitute that. However, the mathematics applied is very sophisticated, and the development of methods applicable in general still in progress.

In the end, I returned to the historic approaches and found them very useful because these reveal the reasons for some effects more vividly than sophisticated methods based on abstract analytic theorems and lemmas. Besides, a non-mathematician alike me rarely knows the names and the thereto related implications of the applied lemmas, rendering methods based thereon a cause for applied mathematics experts.

Concluding, I dare to say that with the help of quite short Markov chains and the matrix geometric and matrix analytic approach most queueing problems can be sufficiently accurate analysed if the initial assumptions and simplifications are wisely chosen. Rubbish results if dull chosen, and without proper care the state transition diagrams may grow huge, too huge to be handled by standard computer systems. Realistically, effective systems are prevalently small, at least if the time basis is correctly chosen and background flows are merged into aggregates as much as possible. Rarely need the involved processes be modelled by Markov chains composed of more than two phases because two are sufficient to precisely match the first three moments for any coefficient of variation $c_X \geq 0.5$. Thus, in most cases we can reduce the problem to two processes with two phases each, and four flows: the flow under test, its peers, the more privileged, and the less privileged. A queueing system based thereon and providing space to hold load equivalent ten times the serving capacity can thus be modelled by a finite state transition diagram, connected by a commonly rather sparse transition matrix, which *Octave* can effectively handle.

Acknowledgements

Foremost my gratitude is dedicated to my advisor, Professor HARMEN R. VAN AS, who not only advised me in the course of preparing and finishing my Ph.D. thesis, he also provided the calm and relaxed working environment necessary to prepare a thesis alike the one I finally managed to finish. In this context many thanks also to the numerous colleagues I met at the institute over the years, who all have influenced the final result in one or another way via the discussions we had in team meetings, during project work, and generally meeting each other from day to day. Lastly in this context but not least, my sincere thanks to Professor KARL GRILL, whom I informed quite late upon my choice for co-advisor and who did a great job in reviewing my work thoroughly in rather short time.

Primarily these thanks are dedicated to the colleagues from the institute of broadband communications (IBK) whom I worked together with. This includs my master thesis advisor Admela Jukan who left for long, but without her having initiated my interest in scientific work, this thesis would not exist. In 2011 the IBK merged with the institute of electronics and information engineering to form the new institute of telecommunications (ITC). In the following years I met a lot of new colleagues from the other part and them I also owe thanks, at least for the social events they organised and the administrative tasks taken over. In this context my deepest thanks to Juliane, the former secretary and administrative mastermind of the IBK, who continued to support me with administrative issues after the merge, as much as still possible. Also thanks to the present administrative staff of the ITC, including the network managers, in particular Matthias, who backed me in keeping my LMDE (Linux Mint Debian) installation with XFCE window manager running past its lifetime. Thanks also to the programmers and communities offering and maintaining *Linux*, *LaTeX*, *Octave*, *Geany*, *Midori*, *Sylpheed*, and all the other tools required day by day, which most of the time worked fluently once I found those that fit my demands.

Thanks evidently and sincerely also to my family and friends. In particular my parents for their love and understanding. They accepted my choice to return to studies after several years in the industry, and I am aware that this step back was much easier for me than for them, wishing me to make the best possible career.

Finally, I want to dedicate this work to my cat, *Kali*, precious in all the term's meanings, that accompanied me for seventeen years, adding a little bit of responsibility to my life. She passed away from old age two years ago, leaving me alone with my unfinished, seemingly endless 24/7 occupation. Ok, she was no big help, but working through weekends without another sensible being around is not the same as in company of a beloved pet.

Vienna, February 2015 Gerald Franzl

Contents

1 Introduction **1**
- 1.1 Communication networks . . . 1
- 1.2 Multi Protocol Label Switching (MPLS) . . . 6
- 1.3 Performance evaluation, analysis, and prediction . . . 11
  - 1.3.1 Performance evaluation methods . . . 14
  - 1.3.2 Performance analysis . . . 15
  - 1.3.3 Performance bounds . . . 17
- 1.4 Markov-chain models . . . 19
  - 1.4.1 Steady state analysis . . . 19
  - 1.4.2 Flow time distribution . . . 22
  - 1.4.3 Solving equilibrium equations in matrix form . . . 23
- 1.5 Distributions and statistical evaluation . . . 28
  - 1.5.1 Basic distributions . . . 28
  - 1.5.2 Statistical values versus moments of distributions . . . 31
  - 1.5.3 Confidence intervals . . . 36
  - 1.5.4 Statistical inference techniques . . . 40
- 1.6 Outline and hypothesis . . . 43
  - 1.6.1 Traffic and system models . . . 43
  - 1.6.2 Traffic handling network elements . . . 44
  - 1.6.3 Operating networks of individual elements . . . 45
  - 1.6.4 An outlook beyond the covered issues . . . 47

2 Traffic and system models **51**
- 2.1 Markovian processes . . . 52
  - 2.1.1 The Poisson process . . . 52
  - 2.1.2 Phase-type distributions . . . 53
  - 2.1.3 Markov Arrival Process – MAP . . . 60
  - 2.1.4 Composed MAP processes and the extension to batch arrivals . . . 62
- 2.2 Non-Markovian processes . . . 67
  - 2.2.1 Deterministic distribution . . . 67
  - 2.2.2 Uniform distribution . . . 69
  - 2.2.3 Heavy tailed distributions . . . 70
  - 2.2.4 Beta distribution . . . 77
  - 2.2.5 General distribution . . . 79
- 2.3 Fitting a distribution to match a sample sequence . . . 80
  - 2.3.1 Comparing of distributions . . . 81
  - 2.3.2 Fitting a negative exponential distribution . . . 84
  - 2.3.3 Moments matching . . . 85
  - 2.3.4 Fitting the H_k and Cox_k model to extreme distributions . . . 86
- 2.4 Traffic flows and network load composition . . . 91

2.4.1 Network load definition . . . 92
2.4.2 Summation of flows toward traffic aggregates . . . 94
2.4.3 Typical traffic flows . . . 96

3 Queueing systems **101**
3.1 Infinite queueing systems . . . 102
3.1.1 *M/M/n* queueing systems . . . 103
3.1.2 Queueing disciplines . . . 107
3.1.3 *M/G/1* queueing systems . . . 114
3.1.4 *GI/M/1* queueing systems . . . 116
3.1.5 *GI/G/1* queueing systems . . . 121
3.1.6 The matrix geometric approach to infinite queues . . . 127
3.2 Finite queueing systems . . . 135
3.2.1 *M/M/n/s* queueing systems . . . 138
3.2.2 Queueing disciplines . . . 142
3.2.3 The matrix analytic approach to *Ph/Ph/1/s* queueing systems . . . 147
3.2.4 Finite population – the *M/M/n/s/c* model . . . 150
3.3 Queue-less systems . . . 155
3.3.1 Loss systems – *M/G/n/n* . . . 155
3.3.2 Infinite capacity – *M/G/∞* . . . 160
3.3.3 Processor sharing – *M/G/PS* . . . 165
3.4 Queueing system simulation . . . 177
3.4.1 Event based simulation and the elimination of absolute time . . . 177
3.4.2 Single server queueing system evaluation by simulation . . . 182
3.4.3 Different queueing disciplines . . . 184
3.4.4 Finite system size introducing blocking . . . 186
3.4.5 Finite customer population - the *Engset setting* . . . 187
3.4.6 Multi servers systems *G/G/n/.*. . . . 188
3.4.7 Processor sharing - *egalitarian* and *discriminatory* PS . . . 190
3.4.8 Implemented *G/G/n/s/X* simulation core and its application . . . 193

4 Traffic management **197**
4.1 Resource sharing . . . 199
4.1.1 Egalitarian resource sharing . . . 199
4.1.2 Traffic prioritisation . . . 205
4.1.3 Weighted scheduling . . . 212
4.2 Congestion management . . . 231
4.2.1 Queue filling based thresholds . . . 232
4.2.2 Early congestion detection . . . 239
4.2.3 Ingress control . . . 247
4.3 End-to-end performance . . . 264
4.3.1 Chained queueing systems . . . 264
4.3.2 Networks of queueing systems . . . 268
4.3.3 Queueing network approximation . . . 274
4.3.4 Quality of service . . . 280
4.4 Future network operation schemes . . . 285
4.4.1 Physical network: resource management . . . 286
4.4.2 Virtual network: consistent capacity management . . . 288
4.4.3 IT-services stratum: application management . . . 291
4.4.4 Lean connectivity provisioning . . . 293

5 Conclusions **295**

A Addenda **299**
A.I Program code 299
A.I.1 Example script (*m*-code) to generate plots 299
A.I.2 *G/G/n/s/c/policy* simulation procedure (*m*-code) 301
A.I.3 Filling and solving the Q-matrix for finite multi-flow systems 310
A.II Event based network simulation 311

Nomenclature **313**
Text styles 313
Variables and operators 313
Abbreviations 313

Bibliography **317**

1 Introduction

Πάντα ῥεῖ (panta rhei) "everything flows"

You cannot step twice into the same river,
for other waters and yet others go ever flowing on.[1,2]

Everything flows and nothing abides. Everything gives way and nothing stays fixed.[2,3]

1.1 Communication networks

Telecommunication has become an integral part of modern life. We rely on the availability of means to communicate over any distance not only in business. Also in our private life we more and more use electronic communication means to inform and organise ourselves. The provided communication services become more and more integrated in daily life, and the variety of *information technology* (IT) services provided and regularly used increases continuously.

Telecommunication, meaning communication over a distance that human voice cannot travel by its own, is not new. First systems that realise telecommunication were signal fires. Today, we literally have access to information from all around the world at our fingertips (figure 1.1). The signal fires

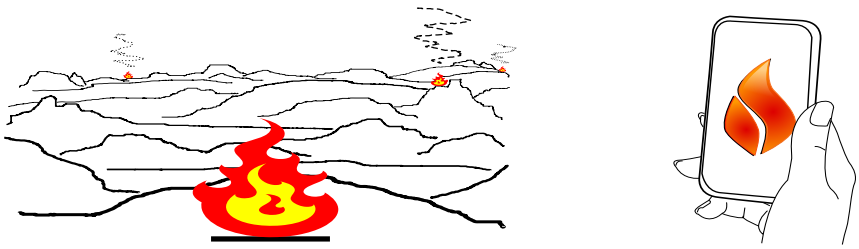


Figure 1.1: From local signal fires to worldwide information sevices

were replaced by antenna masts, and people actively select the source of the information they receive. To realise latter, we need a joint global network that connects all the sources and destinations of information flows. The end-points of this global information exchange system we call *terminals*.

In between these terminals exists an ever-growing, rather invisible, communication infrastructure, which we usually call *the Internet*. This is so complex, that it commonly is sketch as a cloud. Services that cannot be realised without it are accordingly called *cloud services*. Technically, this invisible communication infrastructure comprises a heterogeneous, continually changing conglomerate of mechanisms that *enable, control, and manage the access to, and the delivery of, the information requested by the people via their terminal devices*.

[1] http://en.wikipedia.org/wiki/Heraclitus: DK22B12, in Arius Didymus apud Eusebius, Praeparatio Evangelica, 15.20.2
[2] http://community.middlebury.edu/~harris/Philosophy/Heraclitus.html
[3] http://www.wisdomlib.org/buddhism/book/the-buddhist-philosophy-of-universal-flux/

Connection oriented networks

A first electronic communication system was telegraphy, using for example the Morse code (1836) to transport the information. The telegraphy system relied on point-to-point transportation of intensity modulated analogous signals and relay stations (*nodes*) to route and re-amplify the signals as long as the destination was not reached. The telephony networks used the same principle, modulating the carrier signal by the sound intensity of the spoken words, until the migration to digital transmission. For digital transmission the voice signal has to be sampled, and in between the samples of one communication, now referred to as *call*, the samples of other calls could be transmitted. This method is called *time division multiplexing* (TDM). Thereby the capacity of the medium is split into so called transmission *channels*. However, to establish the required end-to-end connections the switches now also need to de-multiplex the different channels in order to connect them individually. Broadcast networks are a different category. They distribute one signal to all receivers within their coverage area, alike signal fires. In contrast to telephony, broadcast radio prevailingly uses *frequency division multiplexing* (FDM) to transmit different channels over the same medium. In optics FDM is called *wavelength division multiplexing* (WDM). Think of fires that shine in different colours, and the principle should be clear.

Modern communication networks support point-to-point, broadcast and services in between, so called multicast services that connect any group of terminals. Also the medium changed, today most communication networks rely on silica fibre based optical transmission, independent of the last mile technology. Within core networks, sometimes called *backbones*, the *synchronous digital hierarchy* (SDH) or its descendant, the *optical digital hierarchy* (OTH), is used to transmit information between physical locations. OTH is part of the *optical transport network* (OTN) standard, which also integrates WDM for a smart utilization of the silica fibres' tremendous usable capacity in the area of Tbit/s, way too much for a single application, and also too much for today's digital processing. To increase the number of multiplexed channels even further, *code division multiplexing* (CDM) and combinations of multiplexing schemes are today in use to achieve any convenient channel granularity.

Connection less networks

The invention of the *Internet protocol* (IP, 1974) split the world of communication networks into physical and virtual connectivity (figure 1.2). IP uses datagram transmission, being the connection-

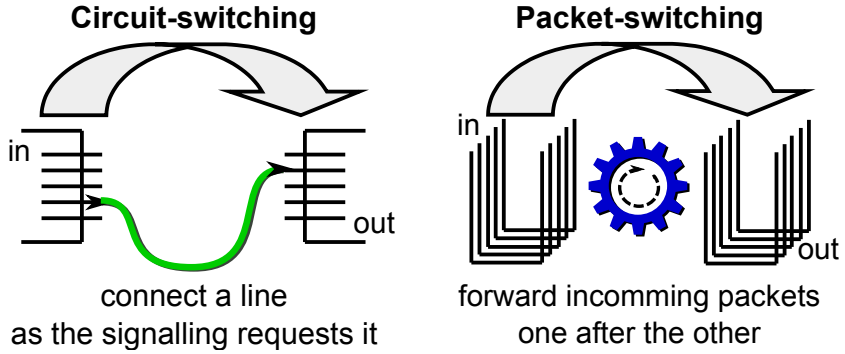


Figure 1.2: Circuit switching and packet switching paradigm

less transport of data by individual packets. Each packet is independently switched at every intermediate node based on some information provided by the packet itself. The key advantage is that the payload of different communications, being the *flow* of packets each creates, autonomously share the available capacity without the need to specify the demand in advance. This capacity sharing scheme is called *statistical multiplexing*.

The switching paradigm is comparable to the common postal service: sort the packets that arrived in ingress-bins to the available egress-bins based on the information shown on the packet's

address-label. Evidently, the packets need to be stored at nodes while the header information, the label, is processed. In addition, if at any time more packets arrive than can be forwarded, the excess packets become buffered. Therefore, the general paradigm is called *store-and-forward* switching, which clearly separates it from *cut-through* switching, the paradigm realised by traditional telephony networks. These two are as different as day and night, cat and dog, or telephony and postal service. Still, they serve the same demand: transportation of information from one location to some other.

Packet routing

Every packet switching node needs a so called *routing table* to determine where a packet shall be forward to. It specifies an output port for every destination address. A routing protocol fills and maintains these tables. Commonly, the routing protocol implements in a distributed way *Dijkstra's algorithm* for finding the shortest paths tree from any node to a destination. The routing tables are constantly maintained in order to respond to resource changes, mainly failure or addition of resources. Persistently defined end-to-end connections do not exist. Particularly not, if load balancing across parallel paths is performed. This *traffic engineering* feature is for example provided by the *open shortest path first* (OSPF) routing protocol, which relies on the *k-shortest paths* Dijkstra algorithm to find up to k parallel paths between any two network nodes.

IP itself does not specify how packets are transmitted, it presumes networks underneath capable to perform this task. This adds essential freedom because it allows IP connections to utilize different transmission systems on a hop-by-hop fashion. The point-to-point connections in between IP nodes can be realised by any transmission system. In the core network this is traditionally a circuit-switched connection provided by a telephony network, but new approaches are emerging. Using *Ethernet* to connect IP nodes adds another statistical-multiplexing layer. However, the network layer of Ethernet is commonly circumvented by static switch configuration: switching tables and port bindings are set by a management system rather than autonomously learned from the traffic, as Ethernet invented it. Such confined, Ethernet offers steady connections between IP nodes. Note, static links are mandatory for reliable IP routing, and therefore no performance advantages can be gained from the added complexity, at least not concerning the end-to-end achievable transport performance.

Distributed control

IP is a fully distributed communication system that operates as long as two nodes are somehow connected, similar to signal fires. It completely separates point-to-point transmission from end-to-end transport, as shown in figure 1.3. Jointly these features cause that IP is very robust, widely applicable,

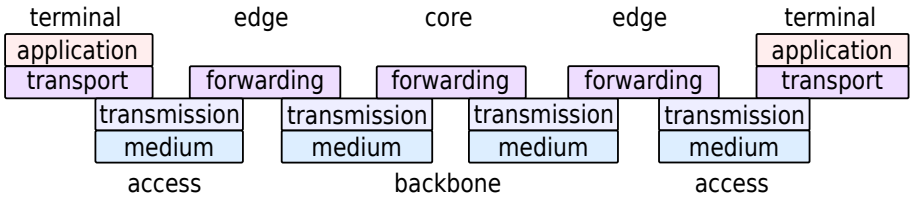


Figure 1.3: Store-and-forward based virtual connection

and extremely simple, basically because no coordination or central management is present. Simple systems nearly always squeeze out less simple, but they may be operationally complex. While with telephony networks the connection set-up mechanism assigns resources to be exclusively used by the requesting connection, IP networks assign the resources packet-by-packet hop-by-hop. Packets are greedy, they want to be forwarded to their destination as fast as possible. Nodes do not coordinate their current routing decisions with downstream nodes. Consequently, fully comparable with car traffic, congestion occurs from time to time. Rarely if the load is low, but with increasing frequency the more utilized the resources are.

The essential companion of IP that takes care of congestion is the *transport control protocol* (TCP) [RFC675, RFC1122]. Initially, IP and TCP are inseparable because TCP was designed to take over all network reliability issues: TCP limits the amount of traffic a source may insert within a certain time-window and dynamically adjusts this limit depending on the success of previous packets. Besides, TCP also assures that every packet is successfully received by re-sending packets that are not acknowledged by the destination. This introduces a tremendous variability of the transport delays, called *jitter*. Therefore, time critical services do not like TCP. They prefer the *user datagram protocol* (UDP) [RFC768], which is very greedy, because it has no mechanism to adjust the inserted load to the current performance of the connection. Alternative protocols that serve time critical transport services more resource friendly have been proposed but seem rarely implemented or used, for example the *Datagram Congestion Control Protocol* [RFC4340]. Most widely used today should be the *Stream Control Transport Protocol* [RFC3286, RFC4960] as it is tailored to streaming services.

Service provisioning

Today, nearly all communication services have been realised over IP. This was triggered by the omnipresence of IP, which results from the flexibility to use any transmission system. Also the simplicity of its management and the continuous speed increase of modern processors supported this move. Merging all networks into one saves operational costs, but if overstretched, the simple workhorse kicks back. Different services cause different loads, as shown in figure 1.4, and demand different transport qualities. If the networks switching paradigm does not support this, either all

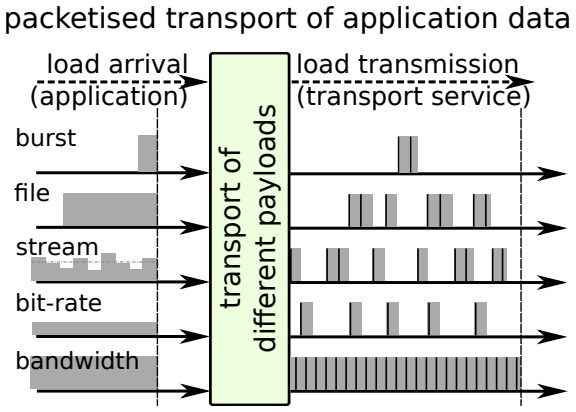


Figure 1.4: Packet flows that services may cause if transported over IP

connection must fulfil every service's quality demand, or add-on mechanisms need to be implemented in order to protect critical services from the effects of congestion. Latter contradicts the network neutrality, being that all packets are handled equally. It also violates the layering defined by the famous Open Systems Interconnection (OSI) model for digital communication systems [1], if the transport of a packet is considered to be the only service provided by the network layer. This violation can be removed, if we allow the transport to be parametrised, for example by adding a label that refers to a *policy* according to which the packet shall be handled at intermediate network nodes, ans shown in figure 1.5. The introduction of the service layer enables the separation of services from their technical realisation. This next important step in the abstraction of network operation is standardised as the so called *next generation network* (NGN) architecture [2]. It postulates that the application characterises the service it requires. These services, and the features thereof, constitute the openly defined *service stratum*. For each service the network shall autonomously selects the means to enable its transport by providing the required resources and mechanisms, which constitute the so called *transport stratum*.

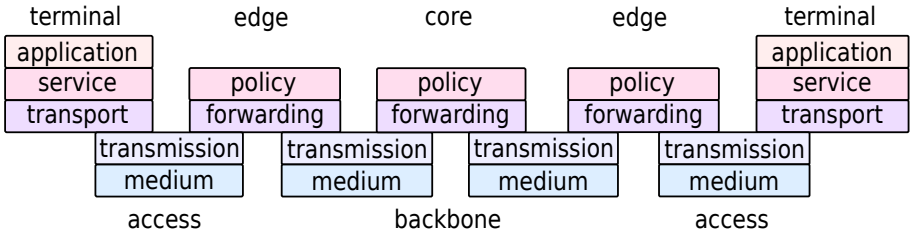


Figure 1.5: Virtual connection using policy based forwarding

End-to-end service quality

Independent of how a service is defined and realised, the performance of an end-to-end connection depends on the current state of the resources that establish the connection. For packet switched connections the primary reason for insufficient performance is temporary congestion. A completely idle network would offer perfect service quality. In practice, congestion deteriorates the performance from time to time and at changing nodes, if the network is well designed and configured, being the core tasks of *network engineering*. However, even the temporary problems along paths minder the average performance, which determines if a connection is useful for a service or not.

Even though all contention related issues and procedures are effective only while there is congestion, they determine the transport quality, called *quality of service* (QoS). The QoS provided by a chain of rather independent network components that constantly change their state, is not easy to determine. Figure 1.6 intends to express this mathematically. Traffic arrivals are modelled by an

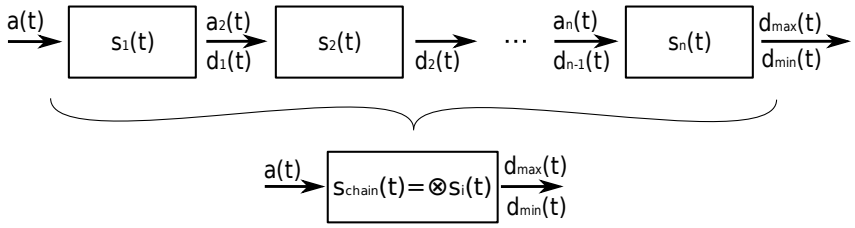


Figure 1.6: Connection quality being a component performance cascade

arrival functions $a_i(t)$, departures from components by departure functions $d_i(t)$. We note that not all departures of the component ahead $d_{i-1}(t)$ need to enter the next component, and vice versa, that also traffic along connections not entirely part of this chain may pass involved components, adding locally the traffic $a_{i,j}(t)$. Thus, $d_{i-1}(t)$ will have some influence on $a_i(t)$, but does not fully define it: $a_i(t) = f\,(d_{i-1}(t), a_{i,j}(t))$. The influence of a component on the traffic passing it is represented by the service function $s_i(t)$, and the connection's global service function by $s_{\text{chain}}(t)$. Note that the time dependence of the service functions follows from their load dependence $s_i(t) = f\,(a_i(t))$, and that the operator '$\otimes$', which relates the global service function with the chained service functions, does not identify a defined operation, it is a place-holder only.

Methods to derive or approximate the connection performance $s_{\text{chain}}(t)$ from the chained component performances $s_i(t)$ are presented in chapter 4.3.4. For now we just list basic relations:

- Increasing the performance of a component will not decrease the QoS of a connection passing it; and increasing the performance of all components will improve the QoS of all connections.
- Privileging the packets of one connection likely deteriorates the performance of all connections that share a resource with the privileged one.
- If all connections are equally privileged they achieve the same performance as without privileging any $\Rightarrow$ the average performance cannot be improved by privileging.

1.2 Multi Protocol Label Switching (MPLS)

Based on the restrictions of plain IP we show what *multi protocol label switching* (MPLS) adds to IP and outline the mechanisms that are required for performance oriented load management and service differentiation. MPLS by itself does not offer QoS provisioning. In the previous section we recognised how a connection's performance depends on the components passed by the flow of packets that belong to a service. We noted that the performance is determined by temporary congestion, which occurs more frequent if resources are more utilized. The relation between the two is progressive, and therefore, the average performance of a network can be improved if peak loads are reduced. This is the core task of *traffic engineering*. In short it means: *put the traffic where the resources are* and you get the best possible network performance.

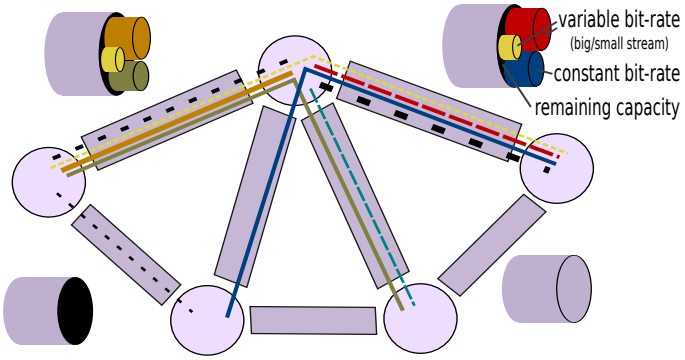


Figure 1.7: IP network with default minimum hop routing

Shortest path routing, being the default IP routing scheme, does not support load balancing or other traffic engineering attempts well. As shown in figure 1.7, the traffic flows may concentrate on links while leaving others underutilized. To increase the performance of an IP network we can adjust the resources or do manual reconfiguring. The first approach follows a rule of thumb: *double the capacity when the utilization reaches a critical level*, for example 50%. This adjustment scheme realises an adaptive one-way optimisation – the demand driven long term evolution of networks. The other method tries to redirect the load by adjusting the link weights used by the routing (minimum cost), as shown in figure 1.8.

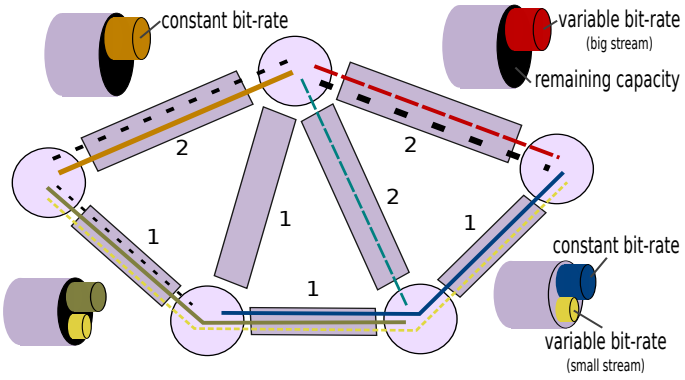


Figure 1.8: IP network with minimum cost routing

Calculating the link weights that provide the best distribution requires good knowledge about traffic flows, not about link loads, and is commonly performed off-line [3]. In addition, once flows toward a destination merged, they cannot be split over different paths any more. The connection less

IP routing can only assign one output port per destination address. Open shortest path first (OSPF) routing offers in its traffic engineering (TE) extension among others the feature to split the load over parallel sections [3, 4]. Commonly it performs this on the aggregated traffic, and this may cause out-of-order packet reception possibly unsuitable for some services.

Both network engineering solutions have network wide side-effects and only indirectly achieve the traffic engineering goal. Let us note here that balanced utilization also can be achieved by routing the traffic over excessive detours. This cannot be the solution. We always should try to minimise the peak utilization and the total resource occupation jointly. Form the connections point of view we want to find a balance between the effectiveness of a longer path consisting of low loaded hops compared to that of a shorter path containing higher loaded hops.

Connection oriented data flows

To enable load management on a per flow basis, and to migrate the proven load balancing schemes from circuit switching, the traffic needs to follow end-to-end defined routes. This is the primary feature that *multi protocol label switching* provides. It separates the transport of payloads form the routing, and makes routing an exchangeable control plane task [5, 6].

The name MPLS consists of two parts: the *label switching* refers to the long known label swapping scheme that scalable implements forwarding relations, and the *multi protocol* term, which expresses that different routing protocols can be used independently, interchangeably, and concurrently to determine the route of a *label switched path* (LSP). The LSPs are set-up and torn-down by signalling comparable with circuit switching. An important difference to other label swapping implementations is: MPLS allows to stack labels, meaning to use existing LSPs alike links and to tunnel new LSPs inside existing ones.

Label swapping is performed to keep the size of labels small and at the same time allow huge numbers of LSPs. Latter is achieved, because lables can be freely reused hop by hop. The label switching is performed based on the (port,label)-tuple: for every ingress-tuple the switching table provides an egress-tuple that specifies the output port and the label to be used on the next hop, as shown in figure 1.9. This mechanism works as good as labels per LSP would, but it reduces the

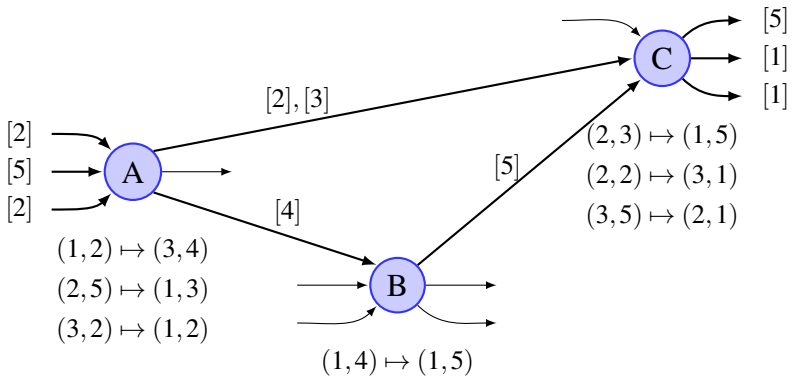


Figure 1.9: Label swapping scheme

number of bits required in the packet header, and, it makes the scheme scalable to any network size. The lable swapping has no effect on the LSP's performance, and therefore we assume this to work and use in the following LSP-IDs to identify different LSPs and not the always changing real labels.

For completeness: nodes that perform label switching are called *label switching router* (LSR), and edge nodes that assign loads to LSPs are called *label edge router* (LER) – this naming is used in the IETF RFC3031 that specifies MPLS [5]. For consistency and ease of reading we hereinafter stick to the general term *node* and presume that the nodes are configured to perform whatever is required at their site.

Traffic load distribution

Load management, meaning advertising the expected load with LSP signalling, keeping track of the load currently occupied on resources, and reserving capacity shares to LSPs, is not part of MPLS. However, to route LSPs that fulfil certain QoS demands, it is essential. For example, OSPF-TE uses this information to route LSPs. Therefore, we assume it, and note that for a signalling based connection set-up and tear-down mechanism it is no challenge to implement load advertisement. A widely used signalling protocol providing the required means is for example RSVP-TE [7]. We thus assume that the expected load per LSP, particularly its mean volume and its distribution, is roughly known at every network node the LSP passes, independent of the signalling protocol used. Form this information we will calculate the congestion potential per outgoing link. We note that not assigned IP traffic either needs to be guessed well or an influence on the LSP's performance technically prevented, to enable trustworthy performance prediction.

Where required we assume in addition to load management a QoS table per node that tells for every ingress tuple the policy to be applied to packets travelling within the according LSP. Thereby we add differentiated QoS to MPLS. Again, being technically no challenge for the signalling, we assume it available as required for the installed service differentiation options. The MPLS header already provides three bits to identify a *forwarding equivalence class* (FEC), comparable to what we know from *asynchronous transport mode* (ATM) and *frame relay* (FR), to support service differentiation among LSPs.

Figure 1.10 shows a small network example that illustrates how link resources may be shared using MPLS in conjunction with resource management. If capacity shares are strictly reserved to

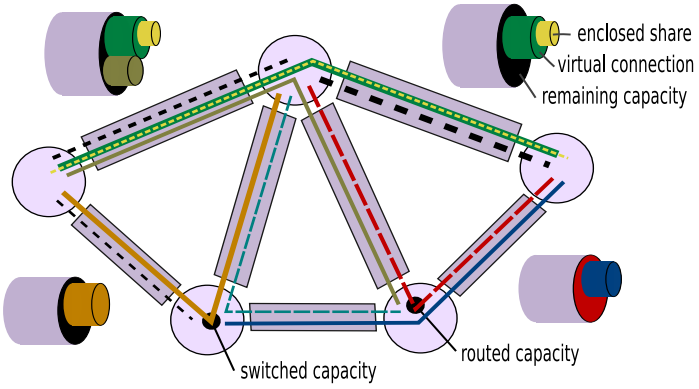


Figure 1.10: Multi protocol label switched (MPLS) network

an LSP, we get a *virtual connection* (VC), because the availability of the demanded capacity share (b_{LSP_i}) is assured to it on every resource along its path. Virtual connections do not need to compete for resources, if overbooking is strictly prevented and no source can insert more than the advertised load b_{LSP_i} at any time.

$$\sum_i b_{\mathrm{LSP}_i \exists_{\mathrm{link}}} \leq C_{\mathrm{link}}, \forall_{\mathrm{links}}$$

More common are LSPs that realise virtual paths. A *virtual path* (VP) has no capacity share reserved to it. Packets flowing along VPs share the currently available resources hop-by-hop. The access to resources can be controlled locally by resource sharing mechanisms, which may implement different policies, called *per hop behaviour* (PHB) in Diff-Serv nomenclature [8]. The difference to IP traffic not enclosed in LSPs is that all packets assigned to a virtual path travel connection oriented along the route of the LSP. The LSP routes may deviate from the shortest path that the not assigned IP packets follow, and LSPs toward the same destination can tunnel the assigned traffic over different, parallel routes to their destination.

We note that the individual packets make use of the currently not used transmission units independent of their belonging to switched or routed LSPs, or no LSP at all. The resource management can realise pre-booking of capacity shares only, not a strict assignment of particular transmission units (timeslots or alike). Therefore, no transmission units remain idle when they may be utilized, and the advantage of statistical multiplexing is fully preserved.

That transmission units are not strictly assigned to virtual circuits introduces some minor jitter, because any packet may need to wait for the next transmission unit in case all are occupied in the moment the packet arrives. Still, it never needs to wait more than the time required for one packet transmission. This jitter component can therefore be neglected. Consequently, hereinafter we concentrate on virtual paths, and assume that the capacity used by virtual connections is never available. The subtraction of the strictly assigned capacity ($C_{\mathrm{VC}} = \sum_i b_{\mathrm{VC}_i \exists_{\mathrm{link}}}$) from the link's raw capacity (C_{link}) assures that we are on the save side: in average the actual performance along virtual paths cannot be worse than the predicted, if we use

$$C = C_{\mathrm{link}} - C_{\mathrm{VC}}$$

as the shareable link capacity for our calculations. Consequently, our results will be far too pessimistic if many unused virtual connections exist. In practice, schemes are required to dynamically release and re-establish virtual connections in order to facilitate best operation of a self-optimising autonomous MPLS regime.

Label distribution

An interesting aspect of MPLS is, at least if the *label distribution protocol* (LDP) [9] is used, that LSPs are set-up from the destination toward the source. Consequently, LSP routes may be changed seamlessly while in operation. The traffic is switched to sections of a new path not before the entire stretch to the destination is ready, as shown in figure 1.11. To set-up a new path or section, first

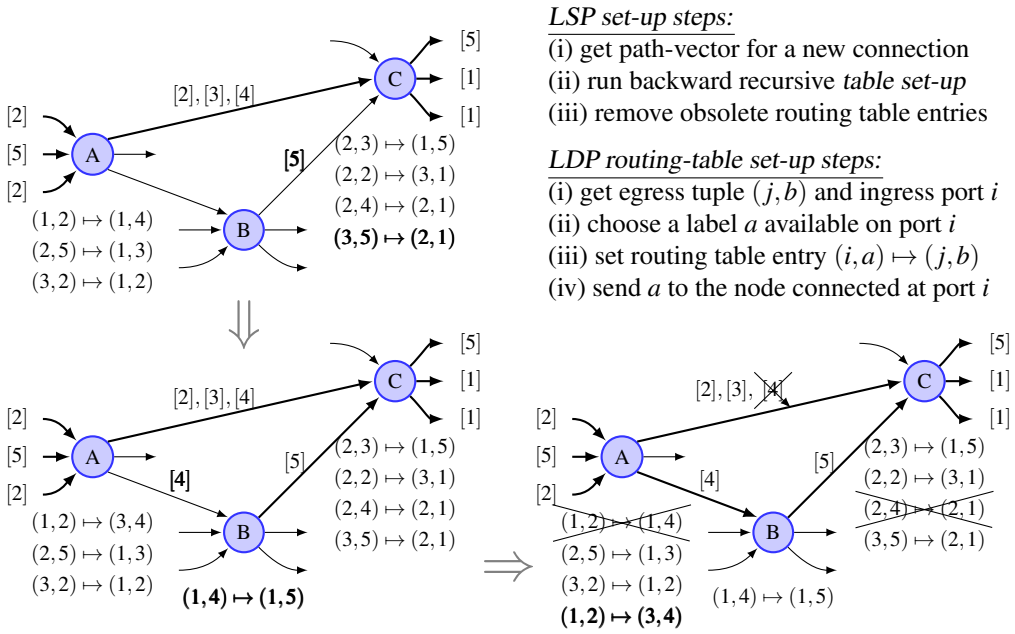


Figure 1.11: Label distribution scheme

a routing protocol needs to be invoked. In case it is successful, the routing returns a path-vector, which contains an ordered list of nodes that uniquely determines the found route. Starting at the destination node the LDP traverses node by node to the source of the new route, and creates new routing-table entries at each passed node. Note that routing-tables must not contain two entries for

the same ingress-tuple. Actually, if this happens, we reached the source and immediately initiate the tear-down of the replaced LSP to remove the obsoleted table entries.

Note that multiple entries with the same egress-tuple are no problem, they just indicate that two LSPs merge in this node. This has to occur while a replacement is set-up in order to assure seamless switch-over (make-before-break), and may be used to define ingress trees that connect many sources with a single destination (e.g., a shared up-link). The destination to source operation enables sectional path restoration, and the seamlessness allows for dynamic load redistribution (in-service optimisation). Both essentially required for autonomous self-organising network control options, including protection strategies and continuous re-optimisation in a dynamic operation regime.

MPLS exemplifying any multi-flow network

Summarising, we see that MPLS enables planned load distribution and supports different per hop behaviours for individual flows. Both introduce the potential for improved end-to-end performance – one on a global scale, the other locally at node sites. This opens a plethora of possibilities, and caused some uncertainties upon its analysis. Today, MPLS is used widely and has proven in practice that it can provide the features required to effectively operate multi-service communication infrastructures. Thus, it would be vain to study the precise protocols that MPLS relies on. To analyse all possible combinations (realisation options) on their QoS delivery potential, is a different story, but excessive. We decided to let aside all implementation details, and thus, skip the analysis of the protocols used by MPLS to realise its features, and in succession, any more detailed presentation and discussion. The interested reader is kindly referred to the IETF RFCs defining MPLS details, extensions and so on, and the many studies on different aspects of MPLS published over the last decade.

We look at the performance MPLS offers from the service side only, and consider the principle functionality, not that of an actual implementation. On one hand, this causes that field experiments would not show the precisely same results. On the other hand, this decision makes the results in a broader sense useful, because MPLS is referred to as example technology only. The functionality required to make our studies applicable solely comprise:

- store-and-forward switching,
- transportation along pre-routed paths,
- expected load advertisement, and
- differentiated load handling options at nodes.

Recently, the enabling principles haven been realised similarly by some Ethernet extensions, and revolutionary network operation scheme that might implement even lower layer switching, for example *optical burst switching* (OBS) and *flow transfer mode* (FTM) [10–13], may appear some time. For all these the principal results shall be applicable accordingly.

Summary of features with particular power and related open issues to be addressed:

- *traffic engineering*: put the traffic where the resources are to globally minimise the peak link utilization. How can this be achieved if traffic appears randomly?
- *service differentiation*: use the LSP identification to assign different per hop behaviours. What are the potentials and merits of different traffic handling strategies?
- *load balancing*: sources may distribute the traffic across parallel LSPs. Can we design a transport control scheme that actively utilizes this option to fulfil QoS targets?
- *distributed optimisation*: LSP routes may be changed while in use. Is a fully distributed management of LSP paths feasible, and how may it perform?

1.3 Performance evaluation, analysis, and prediction

Performance is a very wide and imprecise term if its aspect is not stated. Hereinafter, if not otherwise noted, the term performance refers to the quality of the traffic load transportation, commonly called *quality of service* (QoS). Note that we use the term QoS generally, meaning not bound to any of the different definitions that can be found in the literature. Generally, QoS includes

- the *availability* stating the probability that the service is accessible,
- the *reliability* expressing the likelihood of being flawlessly served,
- the *throughput* being the rate at which load units can be transported,
- the *loss rate* at which load units do not reach their destination, and
- the *transport delay* that the load units experience on their way.

The *jitter*, which is often listed as separate quality metric, relates mathematically to the transport delay, being its first derivative. Because we assume all metrics to be time dependent, it is not listed.

Let us mention instantly, that the momentary throughput equals the rate at which bits are transmitted, and commonly this is a state independent constant rate (*line-speed*). Evidently, this does have a strong impact on the throughput. However, for shared media it cannot serve as performance metric. We have to calculate the throughput as $\frac{n}{\tau(n)}$, where n is the transmitted load volume in *bit*, and $\tau(n)$ is the time in *seconds* needed to transmit this volume. Note that n must be big enough to cover a representative set of load units in order to get the throughput and not the line-rate. When throughput is measured, the opposite approach is often used: the volume $n(t)[bit]$ transported in a fixed time span $\tau[s]$, the *averaging window*, is recorded and the current throughput is calculated as $\frac{n(t)}{\tau}$, as sketched in figure 1.12. This yields results for the recent past only, and τ has to be sufficiently

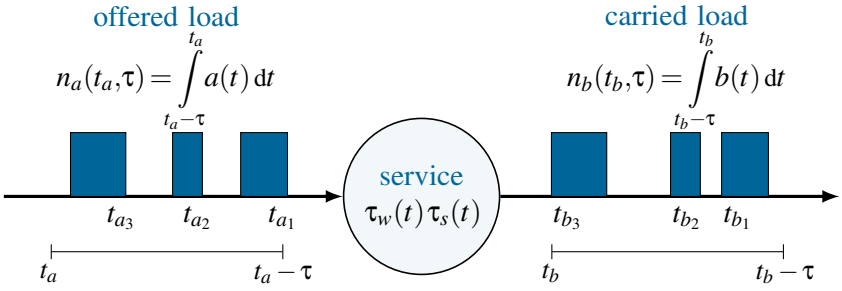


Figure 1.12: Parameters to measure *transport delay*, *throughput*, and *loss rate*

long to get a representative throughput rate. Because the physical unit of a *bit* is [1], bit-per-second equal *Hertz* $=[\frac{1}{s}]$, and therefore is the maximum throughput $\vartheta_{\max}$ sometimes called bandwidth, not to be mistaken with the bandwidth that modulated radio frequencies (RF-channels) or wavelengths (WDM-channels) occupy in the electro-magnetic spectrum.

Figure 1.12 sketches the ingress and egress parameters required to define the transport related performance metrics. With store-and-forward switching experience different load units different serving latencies, *waiting times* $\tau_w(t)$, because of the rapidly changing temporary buffer (*queue*) utilization. Also the *service time* $\tau_s(t)$ may change over time. In particular, it does so, if the system responds to the current load situation when it is *autonomously* or *feedback* controlled. In that case, the currently *provided capacity* $C_s(t)$ becomes time dependent. The *transport delay*, also called *flow time* and therefore abbreviated as τ_f here, is defined as the time in between sending the first bit at t_{a_i} and receiving the last bit at t_{b_i}. We can calculate the current transport delay $\tau_f(t,n_i)$ by

$$\tau_f(t,n_i) = t_{b_i} - t_{a_i} = \tau_w(t) + \tau_s(n_i,t) \tag{1.1}$$

where $\tau_w(t)$ is the time the current load unit has to wait prior its serving starts, and $\tau_s(n_i,t) = \frac{n_i}{C_s(t)}$ is the current *length*, also called *holding time*, of the load unit i with the *size* n_i $[bit]$. We try to

consistently use *length* and *size* accordingly. However, if $C_s(t)$ is actually time dependent, than the length of a load unit with size n_i is time dependent as well, which challenges common notion.

To get the *loss-rate* $n_{loss}(t)$ we need to synchronise $t_a = t_b \to t$ and normalise $n_a(t,\tau)$ and $n_b(t,\tau)$ by the current server capacity $n_s(t,\tau) = \tau C_s(t)$. This yields the normalised rates $n_a(t)$ and $n_b(t)$, and that $n_s(t) = 1$.

$$n_{loss}(t) = n_a(t) - n_b(t) \tag{1.2}$$

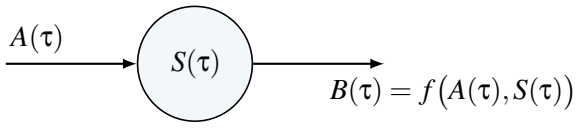
The unit commonly used with $n_a(t)$ is the $[Erlang]$, expressing $n_a(t) \times \mathrm{E}[\tau_s(t)]$. Notably, its unit $[\frac{1 \times time}{time}] = [1]$ is independent of the *time* unit required to determine its components.

Depending on the scope, the same metrics can be defined locally per hop as well as globally per connection. To cover any area, we use the term *network entity*. An entity can be a single component, for example a buffer, or a group of components that jointly realise a functional network element, for example a link between two nodes, or a chain of components and elements that build a functional group, for example a connection. What characterises an entity is a clearly definable functionality.

To validate the usability of a component or connection, being the suitability to serve some application, the *mean values* of QoS metrics, including $\mathrm{E}[\tau_f(t)]$ and $\mathrm{E}[n_{loss}(t)]$, are usually sufficient. Thus, mean values are commonly committed to state the provided QoS. However, to validate the usability for critical applications that cannot compensate variable transport performance, we either need to constantly validate the momentary performance, or more practicably, consider the *variance* or *standard deviation* of relevant metrics in the validation process. For example, the *jitter* may be stated as extra metric to include the second moment of the flow time $\tau_f(t)$.

Modelling the system behaviour

We recognise that performance is not something static. In conjunction with communication services realised by shared resources that transport dynamic traffic loads, this is evident. To state the performance we need to find a method to describe the system behaviour time invariantly. In particular are we interested in a mathematical representation that enables us to calculate the expectable performance for different situations. In other words, we want to *model* the arrival and system behaviour $A(\tau)$ and $S(\tau)$ by mathematical means, such that we can mathematically describe the departure behaviour $B(\tau)$ as a function of the two.



(1.3)

Distribution functions, discussed next in section 1.5, statistically characterise the outcomes of infinite processes, but they do not determine any precise outcome. Thus, a single finite sequence cannot be exactly described by a distribution. However, the characteristics of similar finite sequences, meaning a group that contains potentially infinitely many finite sequence samples, can be precisely characterised by a distribution function. This applies to arrival as well as service functions. In particular, loads can be generally described by an *arrival distribution function* $A(\tau)$, if the underlying process's solution space covers all possible real arrival sequences $a(t)$. And different realisations of a particular network element can be descriptively modelled, meaning generally specified, by a *characteristic service function* $S(\tau)$, if the characteristics of possible realisations $s(t)$ fit those of the process we model the behaviour with.

We try to strictly distinguish between *times* t and *durations* τ as far as possible because this is an important issue in practice. Arrivals and departures happen at time instants t_a, t_b, while inter-arrival and service times are durations τ_a, τ_s (time differences). Further on we use $\tau_x = \mathrm{E}[T_x]$ to refer to the mean of the random variable T_x, which stands for the distribution of the duration $\tau_x(t)$.

It are the durations that determine the transport related performance metrics. For the average service time τ_s this is evident, the smaller τ_s is, the faster is the carried load in average served (*system speed* ς – to be replaced by the service rate shortly).

$$\varsigma[\tfrac{1}{s}] = \frac{1}{\mathrm{E}[T_s]} = \frac{1}{\tau_s} \tag{1.4}$$

The inter-arrival time determines, in conjunction with the system speed ς, the average load ρ,

$$\rho[1] = \frac{\mathrm{E}[T_s]}{\mathrm{E}[T_a]} = \frac{\tau_s}{\tau_a} \tag{1.5}$$

where we recognise that for $\tau_s > \tau_a$, when in average the serving takes more time than passes in between two successive arrivals, we get $\rho > 1$, which indicates an overload situation. Finally, the inverse of the flow time τ_f determines the average throughput ϑ,

$$\vartheta[\tfrac{1}{s}] = \frac{1}{\mathrm{E}[T_f]} = \frac{1}{\tau_f} \tag{1.6}$$

where we notice that service and flow times of lost load units are undefined, $\tau_{s,f}(lost) = no\ number \neq 0$.

Resource utilisation and loss probability

In data networks temporary load peaks exceed the average load considerably. Consequently, we observe the well-known *exponential performance decrease* when the *average offered load* approaches the *provided capacity* [14]. A properly planned data network provides considerably more resources than average load would suggest. The characteristic parameters are:

$$\begin{aligned} \textit{loss factor } p_l &= 1 - \frac{\textit{carried load}}{\textit{offered load}} \\ &= 1 - \frac{\mathrm{E}[n_b(t)]}{\mathrm{E}[n_a(t)]} = \frac{\mathrm{E}[n_a(t) - n_b(t)]}{\mathrm{E}[n_a(t)]} = \frac{\mathrm{E}[n_{loss}(t)]}{\mathrm{E}[n_a(t)]} = \frac{\delta}{\lambda} \end{aligned} \tag{1.7}$$

$$\begin{aligned} \textit{utilisation factor } u &= \frac{\textit{carried load}}{\textit{provided capacity}} \\ &= \frac{\mathrm{E}[n_b(t)]}{C_s} = \mathrm{E}[n_b(t)] \cdot \mathrm{E}[\tau_s(t)] = \frac{\lambda - \delta}{\mu} = \rho\,(1 - p_l) \end{aligned} \tag{1.8}$$

$$\begin{aligned} \textit{over-provisioning factor } o &= \frac{\textit{provided capacity}}{\textit{offered load}} \\ &= \frac{C_s}{\mathrm{E}[n_a(t)]} = \frac{1}{\mathrm{E}[n_a(t)] \cdot \mathrm{E}[\tau_s(t)]} = \frac{\mu}{\lambda} = \frac{1}{\rho} \end{aligned} \tag{1.9}$$

On the far right side we introduce the common mean variables λ, μ, δ, and $\rho = \frac{\lambda}{\mu}$ for mean arrival-, service-, loss-rate, and load, respectively. Note that $\mu = \frac{\varsigma}{\mathrm{E}[n_l]}$ is the system speed normalised by the average size of a load unit, and that we henceforth will use μ instead of ς. If we normalise λ by the average length of a load unit, than becomes $\mu = 1$ and $\rho = \lambda$. However, this is only possible where unique average lengths can be specified. Not used is here the resultant throughput ϑ because it simply results from subtracting the losses δ from the arrivals, $\vartheta = \lambda - \delta$.

On the other side, p_l, u, and o are *factors* — normalised unit-less metrics. The loss- and the utilisation-factor are bound to $[0..1]$. The over-provisioning-factor needs to be greater one to actually represent over-provisioning. For persistent overload (ρ>1) it falls below one, which indicates fatal under-provisioning causing persistent congestion, and thus, disastrous network performance.

For practical reasons the dynamics of the performance metrics are commonly not explicitly stated, as we did in the above eqution. First, because it is cumbersome and excessive to specify distribution functions when mean values are sufficient. Secondly, because only for mean values such simple relations among metrics can be stated. And most restrictively, the precise service distribution provided by network entities cannot be generally defined because the layer abstraction makes the hardware exchangeable. The precise performance in terms of a service function is indeterminable if a specific hardware cannot be presumed.

1.3.1 Performance evaluation methods

The performance of a network entity depends on the traffic $a(t)$ it needs to handle, in particular the present load and its distribution over time. To evaluate the performance of a network entity we have to use statistics, analysis, or numerics, to get results that are generally valid. We use mathematical models (a) to consider the randomness of communication, (b) to avoid singular results, and (c) to derive the typical performance. Thereby, the actual technical realisation is replaced by a more general mathematical characterisation. However, when interpreting evaluation results we need to be aware of the merits of the approach and the mathematics we apply in order to avoid misinterpretations.

Table 1.1 summarise the strengths ($+$) and weaknesses ($-$) of different approaches. The rating depends on their realisation and execution, in particular the availability of the required equipment and skilled personnel to effectively use the available means. For the stated ratings we assumed that both is not a limiting issue. The medium rating ($\circ$) applies where the answer depends on the problem

	experiment	*pure analysis*	*numeric evaluation*	*simulation*
accuracy	$+$	$-$	$\circ$	$+$
reproducibility	$+$	$+$	$+$	$-$
generality	$-$	$+$	$+$	$\circ$
scalability	$-$	$+$	$\circ$	$\circ$
complexity	$\circ$	$-$	$+$	$\circ$

Table 1.1: Performance evaluation methods

size, for example where with unlimited resources the issue would vanish. Reproducibility refers to the likelihood that a different expert team can reproduce the results with their tools.

- The straight approach are *experiments:* Measure the performance observed when the system under test is loaded by different precisely known loads (different traffic traces), and calculate statistics from the outcomes of the performed experiments. An advantage is perfect reproducibility, a core disadvantage is the restriction to statistics over few singular examples. If the outcomes of different experiments vary considerably it is rather impossible to derive generally valid results.

- The most appealing approach is *pure analysis:* Attempt to describe all behaviours by distribution functions: the ingress load by $A(\tau)$, the processing by $B(\tau)$, the time spent in the system by the sojourn time $S(\tau)$, and the departures of the system by $D(\tau)$, and *find the analytic relations* among them. The problem with this approach is mathematical complexity: only for special distribution functions and systems we get closed form representations for $S(\tau)$ and $D(\tau)$. The more $A(\tau)$ and $S(\tau)$ are simplified, the less accurate is $D(\tau)$. However, if we do not know the actual distributions $A(\tau)$ and $S(\tau)$ any reasonable assumption is equally valid and yields reasonably accurate results.

- The most efficient approach is *numerical evaluation:* Model all input and processing by assumed distributions $A(\tau)$ and $B(\tau)$, and asses the characteristics of $S(\tau)$ and $D(\tau)$ via statistically evaluating a numerically calculate set of sufficiently many samples $\{s_i, d_i\}$. Note, the *timing* of results is for steady state analysis irrelevant. This approach merges the strengths and weaknesses of the previous: it removes the restriction to singular examples by analytically describing the input and

system behaviours $A(\tau)$ and $S(\tau)$ via numerically traceable distributions, and the results are statistics, meaning descriptions of the results' distribution, not the distribution itself.

• The always applicable approach is *simulation:* Model all input and system behaviours by software procedures, and get statistical results describing $D(\tau)$ from sufficiently many collected samples $\{d_i\}$. Here, confidence intervals are essential to draw trustworthy conclusions because the entire evaluation is hidden in the software code. As behaviour models we can now use every possible process that can be coded in software. If the procedures for $A(\tau)$, $B(\tau)$ and $S(\tau)$ are accurate, we get perfectly valid results. However, the effort to implement and run a simulation study may be excessive, and an inevitable challenge are seemingly correct results from buggy or inappropriately used software tools.

1.3.2 Performance analysis

We stated above that pure analysis is the most appealing approach. This is driven by the elegance of its result. An equation provides the fastest and maximally precise option to state the input-output relation, all other approaches yield statistical input-output relations only.

Departure distribution via Laplace transforms

The Laplace and Z-transform are well-established mathematical tools in electrical engineering, especially in signal processing and control theory. Solving stochastic and performance related problems in the transform domain enables closed-form solutions for statistical characteristics and distributions of interest. For example, the convolution ($*$) in the time-domain translates to a multiplication ($\times$) in the transformed domain, while the summation remains the same.

$$X_1(t)+X_2(t)+\cdots+X_n(t) \quad \circ\!\!-\!\!\bullet \quad X_1(s)+X_2(s)+\cdots+X_n(s) \tag{1.10}$$

$$Y_1(t)*Y_2(t)*\cdots*Y_n(t) \quad \circ\!\!-\!\!\bullet \quad Y_1(s)\times Y_2(s)\times\cdots\times Y_n(s) \tag{1.11}$$

Thus, if a random process is applied in series upon the results of another, we can replace the unhandy convolution in the time domain by a multiplication in the transformed domain. Thereby the inter-departure time distribution $D(\tau)$ can for example be calculated from the given inter-arrival time function $A(\tau)$ and the sojourn time function $S(\tau)$ describing the system behaviour.

$$D(\tau) = \mathscr{L}^{-1}\Big\{\mathscr{L}\{A(\tau)\}\times\mathscr{L}\{S(\tau)\}\Big\} \tag{1.12}$$

However, *distribution functions are one sided non symmetric functions*. The service time cannot be negative, nor the inter-arrival time. Thus, the use of transforms for distribution functions may differ a little from how we use it with signal processing and control loop evaluation. Anyhow, in [15–17] numerous methodological approaches are presented. The authors moan in [18] that "probabilistic transform methods have been disfavoured by practitioners". They argument that poor knowledge of transform inversions and the daunting complexity of transforms are the reasons. Obviously, working in the transform domain eliminates the relation to reality, and, if we cannot inversely transform the result, there is no reason for doing it. However, recent mathematical publications [19–21] promote practice oriented engineering by mathematical analysis.

Obviously, this is a vibrant research field of *applied mathematics*. However, engineers should invite mathematicians to their research in order to utilize state of the art analytic methods effectively. Presenting real systems in a way that enables a mathematician to recognise the abstract problem he can solve, will still be a challenge, in particular for the *hands-on characters* involved in creative development tasks.

System evaluation via state analysis

The representation of dynamic systems by state diagrams provides a utile and practice oriented analysis approach. If we can tell the performance of a system while it is in a dedicated state, than we can tell something about the system in general, if we know how likely the different states occur. The primary issue is to identify all the N different system states and to calculate their relevance, the so called *state probabilities* $p(i)$.

For clocked systems we can look at the system just prior or immediately after the tick of the clock. In between ticks the system state is undetermined. With some probability the state changes in between two successive clock ticks to some other state. This probability defines the *transition probability* υ_{ij} for the transition from state i to state j. If we include the transition to itself, υ_{ii}, than in steady state $\sum_{j=0}^{N} \upsilon_{ij} = 1$ and $\dot{p}(i) = 0$ must hold $\forall_i$. The first equality tells that the system either changes its state or remains in the current state, the other that the system probabilities do not change over time (system states are steady). The probabilities for transitions υ_{ij} from one to some other state are determined by external parameters alike arrival-rate and service-rate, such that we commonly can calculate them. These given, we can set up all equilibrium equations,

$$\sum_{j=0}^{N} \upsilon_{ji} p(j) = p(i) \sum_{j=0}^{N} \upsilon_{ij} \qquad \forall i \tag{1.13}$$

stating that in the steady state the probability to enter a state must equal the probability to leave that state, and can solve the cyclic dependent equation system, replacing one equation by the evident condition

$$\sum_{i=0}^{N} p(i) = 1 \tag{1.14}$$

that the sum over all state probabilities must be one, stating that the system has to be in a defined state at any time. However, if we calculated all υ_{ii}, than we can as well calculate the state probabilities directly,

$$p(i) = \frac{\upsilon_{ii}}{\sum_{i=0}^{N} \upsilon_{ii}} \qquad \forall i \tag{1.15}$$

stating that the normalised probability to remain in state i equals the probability to be in that state.

For time continuous systems we replace the transition probabilities υ by *transition rates* r, which reflect the transition probabilities per time unit. The system is now observed whenever something changes, an *event* occurs, just prior the change or immediately after it. Thereby we assure that every possible change is observed, combined events do practically not exist on a continuous, infinitely dense, time axis. Only the steady state equilibrium equations look a little different because the rates replace now the probabilities.

$$\sum_{j=0}^{N} r_{ji} p(j) = p(i) \sum_{j=0}^{N} r_{ij} \qquad \forall i \tag{1.16}$$

As before, the equilibrium equations defined by equation 1.16 state that in the steady state the total rate at which a state is reached must equal the total rate at which it is left. Again, the *state probabilities* $p(i)$ equal the times that the system in average remains in a state, and thus, the state probabilities $p(i)$ tell the fraction of time that the system is in average in a certain state, and thereby its likelihood and relevance.

The state probabilities $p(i)$ provide the weighting factors with which we need to consider the state dependent performance metrics when summing them up to get general, state independent, performance metrics.

$$metric = \sum_{i=0}^{N} p(i) \times metric(i) \tag{1.17}$$

From system models to state transition diagrams

The states that we need to identify depend on the modelled system. In buffer-less systems the states commonly refer to the number of channels (*servers*) currently occupied. For systems with buffers (*queues*) also the number of currently queued customers separates states. And if there are m queues, than every m-dimensional tuple defines a state. Models with n servers and m queues can be designed similarly. Single and multi-queue examples are discussed in chapter 3 and section 4.1, respectively.

The two basic systems, the single queue single server queueing system and the n-server loss system shown in figure 1.13, are briefly sketched and discussed to introduce the basic modelling issues. How to solve these follows in section 1.4. This system sketch needs to be extended to *state*

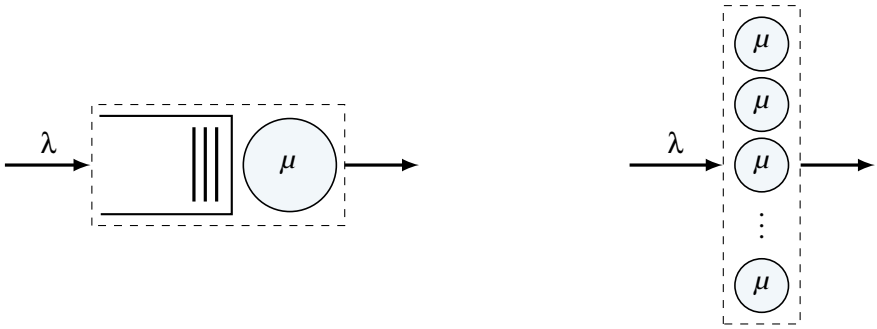


Figure 1.13: Single server queueing system and n-server loss system

transition diagrams, where we assume for the queueing system an infinite queue size, and for the parallel server system a finite number of servers. Latter arises for example with circuit switched communication services (plain old telephony), the former we know from queues in front of a cashier's desk and applies likewise for store-and-forward based packet switches. The according state transition diagrams are shown in figure 1.14, where we marked the states by their probability $p(i)$. Later on

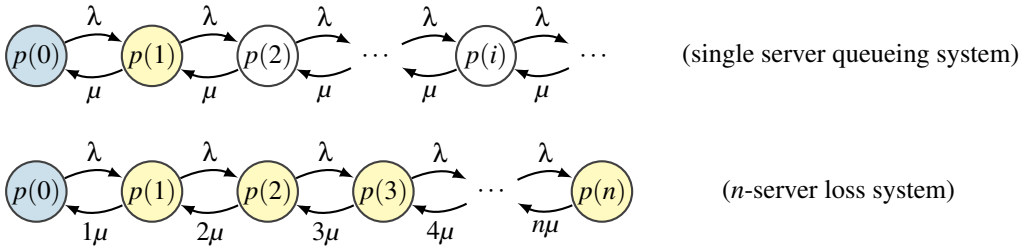


Figure 1.14: Basic state transition diagrams

we will commonly index states by the m-dimensional *tuple* $\langle i,j,k,\ldots\rangle^{[m]}$ that identifies the state. How the states are positioned in the state transition diagram is mathematically irrelevant, only the transitions among them need to be correct. However, a clear and self-explanatory system state ordering supports the usability.

1.3.3 Performance bounds

Network calculus [22, 23] provides the means to handle complex end-to-end performance metrics using cumulative envelopes. There exist two theory strands, deterministic and stochastic [24, 25]. The former yields reliable performance bounds, the latter likely performance regions. Thus, network calculus would provide the means needed to asses the expactable performance by minima or likelihoods, respectively. However, latter is still under development, solutions for some special cases have been found, but no general theory yet.

Network calculus is based on *cumulative distribution envelopes* $a(\tau)$ and $s(\tau)$ for the arrival and service process. These envelopes state upper/lower limits on the load that may arrive/pass in any time interval of length τ. To use these, network calculus applies the *min-plus algebra*,

$$a(\tau) \otimes s(\tau) = \inf_{0 \le t \le \tau} \{a(\tau - t) + s(t)\} \tag{1.18}$$

$$a(\tau) \oslash s(\tau) = \sup_{t \ge 0} \{a(\tau + t) - s(t)\} \tag{1.19}$$

where the *infimum* and the *supremum* functions are the point-wise *min* and *max* operations,

$$d^-(\tau) = \inf_{0 \le t \le \tau} \{a(\tau - t) + s(t)\} = \min_{0 \le t \le \tau} \{a(\tau - t) + s(t)\} \quad \text{performed } \forall \tau \tag{1.20}$$

$$d^+(\tau) = \sup_{t \ge 0} \{a(\tau + t) - s(t)\} = \max_{t \ge 0} \{a(\tau + t) - s(t)\} \quad \text{performed } \forall \tau \tag{1.21}$$

which yield cumulative min/max departure envelopes. Obviously, these can be calculated only numerically. The t in these equations is no time instant, it is the internal running *offset* from τ only.

Applied for network elements, the lower and upper departure envelopes of $d(\tau)$ can be calculated via the *infimum* and the *supremum* of the arrival envelope $a(\tau)$ and the service envelope $s(\tau)$.

$$a(\tau) \otimes s(\tau) \quad \le \quad d(\tau) \quad \le \quad a(\tau) \oslash s(\tau) \tag{1.22}$$

The calculated envelopes given at the left and right side of equation 1.22 bind the maximum and minimum throughput. They specify the worst case load a destination terminal needs to handle and the least end-to-end performance, respectively. Latter is essential to grant that a certain service can be supported reliably.

Super-positioning of arrival envelopes becomes a rather simple summation and cascades of service envelopes can be replaced by the chain's infimum of the individual service envelopes.

$$a(\tau) = \sum a_i(\tau) \tag{1.23}$$

$$s(\tau) = s_1(\tau) \otimes s_2(\tau) \otimes s_3(\tau) \otimes \cdots \tag{1.24}$$

The subtraction of an arrival envelope from a service envelope yields the *leftover* service envelope for other arrivals as it for example occurs with strict prioritisation of signalling traffic.

$$d_1(\tau) \ge a_1(\tau) \otimes (s(\tau) - a_2(\tau))^+ \tag{1.25}$$

Finally, we get for the delay envelope $\tau_f(\tau)$ and the backlog envelope $x(\tau)$

$$\tau_f(\tau) = \inf\{t \ge 0 \mid a(\tau - t) \le d(\tau)\} \tag{1.26}$$

$$x(\tau) = a(\tau) - d(\tau) \le a(\tau) \oslash s(0) \tag{1.27}$$

where $a(\tau - t) = d(\tau)$ specifies the horizontal distance in between the envelopes, and $a(\tau) - d(\tau)$ the vertical distance. The calculation of equation 1.26 is to find $\forall \tau$ the smallest $t \ge 0$ for which the condition $a(\tau - t) \le d(\tau)$ is fulfilled. The $s(0)$ in equation 1.27 refers to the minimum instantaneous service rate, which commonly is zero, such that $a(\tau) \oslash s(0)$ becomes the maximum of $a(\tau)$ over τ. The maxima of these envelopes can be used to dimension network elements to guarantee bounded delay and no losses. However, in general they are too conservative to be economic.

In contrast to the state space based analysis methods, network calculus cannot be used with unbounded distributions because for these no envelopes exist. In theory a restriction it fits real systems: practical needs commonly cause bounded characteristics. Still, it complicates the combination of methods because unbounded distributions alike the negative exponential distribution are commonly used to model the customers' behaviour.

1.4 Markov-chain models

Markov chains are widely used to model dynamic systems with state dependent performance. These states need to be identified, and transition-rates or -probabilities assigned for all the transitions that may occur. The time the system remains in a certain state has to be negative exponentially distributed and thus Markovian, meaning memoryless. Being memoryless yields that transitions out of a state do not depend on how the state has been reached, and this is a mandatory prerequisite for the solving methods discussed.

Depending on the modelled system we identify different states that need to be considered. Bufferless systems provide no queues and thus we need to model the server occupation only. With buffers (*queues*) the number of currently queued customers is an important system state parameter. If there are m logically separated queues, than every m-dimensional queue filling tuple defines a dedicated state. If the service times of customers is not unique, meaning if different traffic classes show individual service times, than also the type of client currently served needs to be modelled, and every server occupation possibility (n-dimensional tuple in case of n servers) defines a separately required state. The two basic systems, single queue queueing system and n-server loss system have been sketched in the previous section 1.3, repeated here in figure 1.15 for convenience. The representation

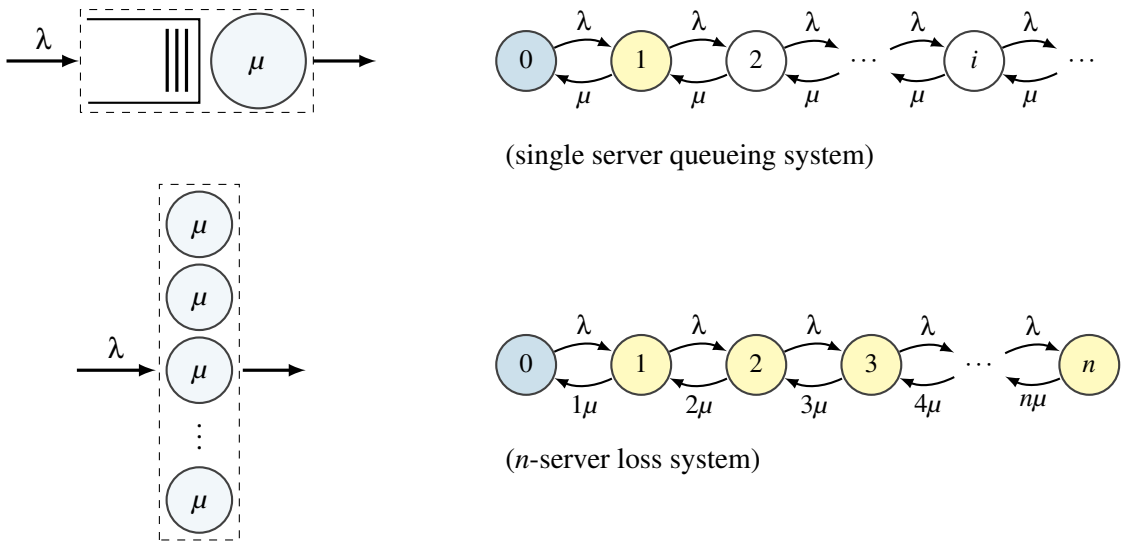


Figure 1.15: M/M/1 queueing system and M/M/n/n loss system

of these systems by *state transition diagrams* has been introduced in section 1.3, figure 1.14. In the following we show how these systems are mathematically solved and how we can derive performance metrics from state probabilities, being the steady state solution of state transition diagrams. If we are interested in the transient behaviour of such systems we similarly can set up a system of differential equations and solve that to get the state transients $\dot{p}(i)$.

1.4.1 Steady state analysis

Solving the system in steady state yields the performance metrics that identify the in average expectable performance. To be steady, the probability to enter a state needs to equal the probability to leave the state. This simple condition yields the state related equilibrium equations. The condition can be extended to any boundary in between two groups of states that in total contain all states, as shown in figures 1.16 and 1.17. Together with the condition that the system needs to be in a defined state at any time (equations 1.30 and 1.41), an equation system results that can be solved [14, 26] to get all the state probabilities, as shown in the following for the exemplary systems.

Single server queueing system

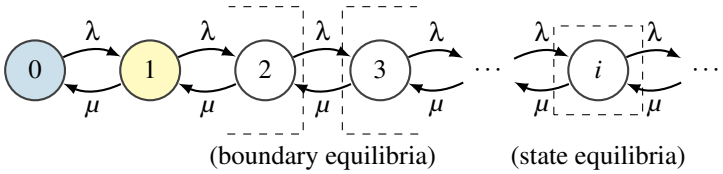


Figure 1.16: Steady state equilibria for the M/M/1 queueing system

To solve the *M/M/1 queuing system* we get according to figure 1.16

$$\begin{aligned} p(i-1)\lambda + p(i+1)\mu &= p(i)\lambda + p(i)\mu && (1.28)\\ p(i-1)\lambda &= p(i)\mu && (1.29)\\ \sum_i p(i) &= 1 && (1.30) \end{aligned}$$

where equations 1.28 and 1.29 result for the two sketched options, state equilibria and boundary equilibria, respectively. For this simple system equation 1.29 readily yields the known recursive equation

$$p(i) = \tfrac{\lambda}{\mu} p(i-1) = \rho p(i-1) = \rho^i p(0) \tag{1.31}$$

that defines the infinite many state probabilities based on $p(0)$, the probability that the system is idle. Inserting equation 1.31 in 1.30 we get

$$p(0)\sum_{i=0}^{\infty}\rho^i = p(0)\tfrac{1}{1-\rho} = 1 \qquad \Rightarrow \qquad p(0) = 1-\rho \tag{1.32}$$

and finally,

$$p(i) = \rho^i(1-\rho) \tag{1.33}$$

the geometrically distributed state probabilities of the infinite M/M/1 queueing system. Note that the infinite system must not be overloaded, the equations demand a load $\rho < 1$.

Using the state probabilities we can calculate the average system filling $\mathrm{E}[X]$ and queue filling $\mathrm{E}[Q]$.[4] Applying Little's law $N = \lambda T$, the mean flow time τ_f and waiting time τ_w follow.

$$\mathrm{E}[X] = \bar{x} = \sum_{i=0}^{\infty} i\,p(i) = (1-\rho)\sum_{i=1}^{\infty} i\rho^i = \frac{\rho}{1-\rho} \tag{1.34}$$

$$\mathrm{E}[Q] = \bar{q} = \sum_{i=2}^{\infty} (i-1)p(i) = \rho(1-\rho)\sum_{i=1}^{\infty} i\rho^i = \frac{\rho^2}{1-\rho} \tag{1.35}$$

$$\mathrm{E}[T_F] = \tau_f = \frac{\mathrm{E}[X]}{\lambda} = \frac{1}{\mu(1-\rho)} \tag{1.36}$$

$$\mathrm{E}[T_W] = \tau_w = \frac{\mathrm{E}[Q]}{\lambda} = \frac{\rho}{\mu(1-\rho)} \tag{1.37}$$

Being an infinite system no losses can occur. Consequently is the departure rate (the throughput) equal the arrival rate $\vartheta = \lambda$. Only the distribution over time and therefore the higher moments change. The *utilization* is the proportion the server is not idle, and thus it can be directly calculated from $p(0)$.

$$u = 1 - p(0) = \rho \tag{1.38}$$

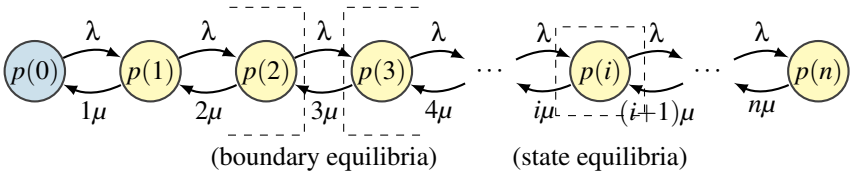


Figure 1.17: Steady state equilibria for the M/M/n/n loss system

Multi server loss system

To solve the *M/M/n/n loss system* we get according to figure 1.17

$$p(i-1)\lambda + p(i+1)\,(i+1)\mu = p(i)\lambda + p(i)\,i\mu \tag{1.39}$$
$$p(i-1)\lambda = p(i)\,i\mu \tag{1.40}$$
$$\sum_i p(i) = 1 \tag{1.41}$$

where from equation 1.40 we find

$$p(i-1)\lambda = p(i)\,i\mu \qquad \Rightarrow \qquad p(i) = \tfrac{\rho}{i}p(i-1) = \tfrac{\rho^i}{i!}p(0) \tag{1.42}$$

and when inserting this in equation 1.41

$$p(0)\sum_{i=0}^{n}\tfrac{\rho^i}{i!} = 1 \qquad \Rightarrow \qquad p(0) = \frac{1}{\sum_{i=0}^{n}\frac{\rho^i}{i!}} \tag{1.43}$$

we finally get the state probabilities $p(i)$.

$$p(i) = \frac{\frac{\rho^i}{i!}}{\sum_{i=0}^{n}\frac{\rho^i}{i!}} \tag{1.44}$$

Therefrom we can again calculate the average system filling $\mathrm{E}[X]$. Not having a queue, the mean flow time must be the inverse service rate $\frac{1}{\mu}$, independent of n. Queue filling $\mathrm{E}[Q]$ and waiting time τ_w do not exist.

$$\mathrm{E}[X] = \bar{x} = \sum_{i=0}^{n} i p(i) = \frac{\sum_{i=1}^{n}\frac{\rho^i}{(i-1)!}}{\sum_{i=0}^{n}\frac{\rho^i}{i!}} = \frac{\rho\sum_{i=0}^{n-1}\frac{\rho^i}{i!}}{\sum_{i=0}^{n}\frac{\rho^i}{i!}} = \rho\,(1-p(n)) \tag{1.45}$$

$$\mathrm{E}[T_F] = \tau_f = \frac{1}{\mu} \tag{1.46}$$

Being finite, this system is stable for any load ρ because what it cannot handle is simply blocked away. The probability that this happens is called *blocking probability* P_b, and it equals the probability that the system is full when an arrival occurs. For Poisson arrivals this equals the state probability $p(n)$.

$$p(n) = P_b = \frac{\frac{\rho^n}{n!}}{\sum_{i=0}^{n}\frac{\rho^i}{i!}} \tag{1.47}$$

This equation is widely known under the name *Erlang B formula*, sometimes without the B, for its importance. The blocking rate, being the average number of blocked arrivals, is $\delta = \lambda P_b$, and the *throughput* ϑ is the arriving minus the blocked load

$$\vartheta = \lambda - \delta = \lambda\,(1-P_b) \tag{1.48}$$

[4]Repeatedly we use $\sum_{i=0}^{\infty}x^i = \frac{1}{1-x}$, $\sum_{i=1}^{\infty}x^i = \frac{x}{1-x}$, and $\sum_{i=0|1}^{\infty} i\,x^i = \frac{x}{(1-x)^2}$, $\forall_{x<1}$, here and henceforth.

where we notice the evident relation between throughput and system filling $\vartheta = \bar{x}\mu$. This results from the property that the average amount served, which occupies a server processing it with μ, must in average also leave the system for stability reasons, at least in the steady state. The *utilization* is the proportion at which the servers are busy

$$u = \frac{\mathrm{E}[X]}{n} = \frac{\lambda(1-P_b)}{n\mu} = \frac{\vartheta}{n\mu} \tag{1.49}$$

where $n\mu$ is the total serving capacity C_s provided by the *M/M/n/n*-system.

Without prove let us mention here that the above stated equations hold for any service distribution, meaning for any *M/G/n/n* loss system. Actually, the method to model dynamic systems by state transition diagram and to solve these in steady state using state and boundary equilibrium equations is not bound to negative exponentially distributed arrival and service times. However, the system needs to be memoryless, meaning that the transition rates out of a state may not depend on how the current state has been reached. And therefore, only the sojourn times of the individual system states need to be negative exponentially distributed.

1.4.2 Flow time distribution

To assess the *jitter*, being the variance of the flow time, we need to know at least the second moment of the flow time distribution F_{T_f}. The steady state solution of the state transition diagram shown above yields only mean values. To get higher moments we either need to set up the differential equations for states and boundaries and solve the resulting *system of differential equations*, or use a different approach. Closely related to the state transition diagram are the *flow diagrams*. Figure 1.18 depicts the flow diagram of the *M/M/1* queueing system. Such a state flow graph enable the direct calculation

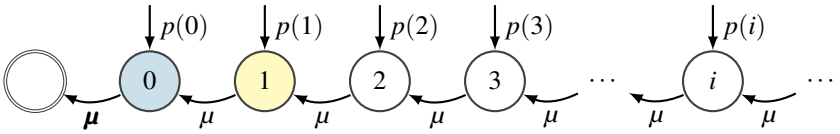


Figure 1.18: Flow diagram for the M/M/1 queueing system with FIFO queueing discipline

of the flow-times that different customers in average experience, given the probabilities at which arriving customers enter the system in a given state are known, for example form solving the state transition diagram. For Poisson arrivals these entry probabilities equal the steady state probabilities. From its entry point the test customer than pass several states until ultimately he is served and leaves the system, entering a so called absorbing state ◎.

The number of states that need to be passed in between arriving and departing depends on the queueing discipline. Here, we stick to the common *first in first out* FIFO queueing discipline, which is also known as *first come first served* FCFS serving discipline. Other notable queueing disciplines include *last in first out* LIFO and *random queueing* (RAND), and are detailed in chapter 3.

Here, with FIFO queueing, a test customer that arrives while the system is in state 2 becomes the third customer in the system and has to wait until the two in front of him have been served. The customer leaves the system after three service times[5]. We will see in section 1.5 that three equal negative exponentially distributed phases in series cause an Erlang$^{[3]}$ distribution. How to model such is shown in chapter 2. Similarly, Erlang$^{[i+1]}$ distributions result for arrivals to states i. This observation lets us already assume that the departure distribution during busy intervals, meaning while customers are in the system, is never more peaked (less smooth) than the service distribution.

[5]Being memoryless the time passed since serving started does not influence the residual time till service completion.

To get the general flow time distribution's *pdf* and its mean we need to calculate the sum over all entry possibilities, weighted by their occurrence probability

$$\tau_f(\tau) = \sum_{i=1}^{\infty} p(i)\,\tau_f(i,\tau) \qquad \forall \tau \in \mathbb{R}^+ \tag{1.50}$$

$$\tau_f = \int_0^{\infty} \tau \sum_{i=1}^{\infty} p(i)\,\tau_f(i,\tau)\,\mathrm{d}\tau = \sum_{i=1}^{\infty} p(i)\,\tau_f(i) = \tau_s \sum_{i=1}^{\infty} i\,p(i) = \tau_s\,\mathrm{E}[X] \tag{1.51}$$

where $\tau_f(\tau)$ is the *pdf* of the general flow time, τ_f its mean, and $\tau_f(i,\tau)$ the *pdf* of the flow time for customers that arrive to the system while it is in the state i, with mean $\tau_f(i) = \int \tau\,\tau_f(i,\tau)\,\mathrm{d}\tau = i\,\tau_s$, where τ_s is the mean serving time (holding time of the server).

The summation of the different distribution functions $\tau_f(i,\tau)$ in equation 1.50 is not trivial. Either we transform all into the Laplace domain, where summation is no problem, or we use point-wise summation of the *pdf*'s to get the resultant *pdf* curve, which visually describes $\tau_f(\tau)$, and allows to numerically calculate its approximate mean τ_f. For an infinite system both is a challenge. In any case, although the individual $\tau_f(i,\tau)$ depend on the service rate μ only, the mean τ_f as well as the gross distribution of the flow time $\tau_f(\tau)$ are load dependent. This is because of the load dependence of the entry probabilities $p(i)$. We can depict the *pdf* or *cdf* in two dimensions only for chosen loads, as shown in figure 1.19.

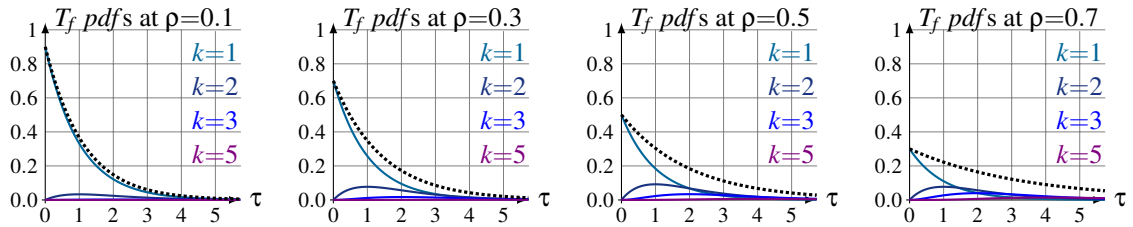


Figure 1.19: Weighted flow time sub-*pdf*s and dotted the calculated total *pdf*(T_f)

Intuitively we can say that for very low loads the flow time distribution will approach the service time distribution, as long as the influence of the waiting time is negligible. For very high loads the mean number of phases to pass approaches infinity, and because $\mathrm{Erlang}^{[\infty]}(\mu) = \mathrm{D}(\infty)$, the flow time distribution $F_{T_f}(\tau)$ becomes approximately deterministic, though at the same time approximating infinite mean, $\tau_f \to \infty$.

1.4.3 Solving equilibrium equations in matrix form

We have to solve an equation system with $n{+}1$ variables, here state probabilities p_i with $i = 0..n$, defined by $n{+}1$ equations and $(n{+}1)^2$ associated parameters q_{ij}. If we move all state probabilities to the left side, the right sides of the equilibrium equations become zero. We thus can represent the equation system in matrix form by

$$Q \cdot p = b \quad \Leftrightarrow \quad \begin{bmatrix} q_{00} & q_{10} & \cdots & q_{n0} \\ q_{01} & q_{11} & \cdots & q_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ q_{0n} & q_{1n} & \cdots & q_{nn} \end{bmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{1.52}$$

where $\forall_{i \neq j}\, q_{ij}$ is the inbound transition rate from state-i to state-j, and because negative rates do not exist, these are all positive. The q_{jj} we find from the condition that $\sum_i q_{ij} = 0 \;\; \forall_j$ to be negative, because for equilibrium q_{jj} must equal the negative sum of all outbound rates.

$$q_{jj} = -\sum_{i \neq j} q_{ij} \qquad \forall_j \tag{1.53}$$

This equation system cannot be solved directly because the equations system is cyclic dependent. We need to replace one equation by $\sum_i p_i = 1$ (equation 1.30). Commonly we replace the last equation, which means that we set all $q_{i,n} = 1$ and also $b_n = 1$:

$$Q^* \cdot p = b^* \quad \Leftrightarrow \quad \begin{bmatrix} q_{00} & q_{10} & \dots & q_{n0} \\ q_{01} & q_{11} & \dots & q_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \tag{1.54}$$

This linear system of equations now can be solved methodically, for example using optimised computation algorithms.

Often the matrix form is specified using row-vectors p^T and b^T. In this case we get

$$p^T \cdot Q^T = b^T \quad \text{and} \quad q_{ii} = -\sum_{i \neq j} q_{ij} \quad \forall_i \tag{1.55}$$

which is more convenient for programming when the first index indicates the row. Consequently, we have to replace the last column in Q^T by ones to integrate $\sum p_i = 1$:

$$p^T \cdot Q^{*T} = b^{*T} \quad \Leftrightarrow \quad \begin{pmatrix} p_0 & p_1 & \dots & p_n \end{pmatrix} \begin{bmatrix} q_{00} & q_{01} & \dots & 1 \\ q_{10} & q_{11} & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ q_{n0} & q_{n1} & \dots & 1 \end{bmatrix} = \begin{pmatrix} 0 & 0 & \dots & 1 \end{pmatrix} \tag{1.56}$$

This form is solved equally, and yields the same results for the state probabilities p_i.

Later on we show how the Q-matrix can be constructed from sub-matrices that represent adjacent regions of the state transition diagram. This method is of particular interest for solving systems that are modelled by *phase type distributions* because these are themselves specified by matrices, as shown in chapter 2.

To exemplify the numeric method we discuss a small *M/M/1/s* example with finite queue and a server that is not always available, as depicted in figure 1.20. Arrivals occur according to a Poisson

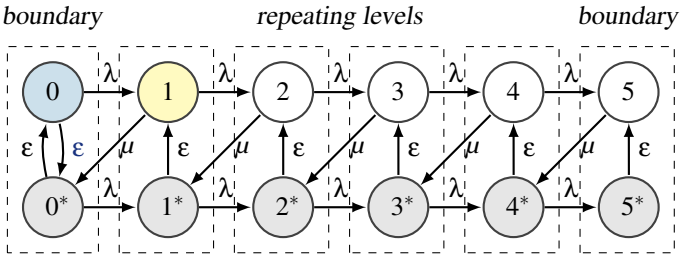


Figure 1.20: State diagram representing a queueing system with vacation

process, negative exponentially distributed with the mean arrival rate λ (mean inter-arrival time $\tau_a = \frac{1}{\lambda}$). The time required to serve a client is also negative exponentially distributed with an average serving time being $\tau_s = \frac{1}{\mu}$. After finishing a job the server stops serving the queue for an negative exponentially distributed time period with a mean vacation duration of $\tau_v = \frac{1}{\varepsilon}$.

To represent this system in matrix form we grouped the states with the same number of waiting customers, as indicated in figure 1.20, and see that similar groups of two states each result. The most left and right group define the *boundary levels* $B(0)$ and $B(s)$. Those in between have identical relations internally $A(\text{internal})$ and in between each other $A(\text{up})$ and $A(\text{down})$, thus they define *repeating levels*. The relations per *level* can be describe by small sub-matrices. For matrices B the

indices refer to the related levels, meaning that at the left border B_{00} refers to *internal*, B_{01} refers to *out=up*, and B_{10} refers to *in=down*. For the levels in between we use the somewhat different indexation where A_1 refers to *internal*, A_0 refers to *up*, and A_2 refers to *down* [27].

$$\begin{array}{ccccccc} & \textit{internal} & & & \textit{up} & & \textit{down} \\ B_{00} = \begin{bmatrix} 0 & \varepsilon \\ \varepsilon & 0 \end{bmatrix}, & & B_{01} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}, & & B_{10} = \begin{bmatrix} 0 & \mu \\ 0 & 0 \end{bmatrix} \\ A_1 = \begin{bmatrix} 0 & 0 \\ \varepsilon & 0 \end{bmatrix}, & & A_0 = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}, & & A_2 = \begin{bmatrix} 0 & \mu \\ 0 & 0 \end{bmatrix} \\ B_{s,s} = \begin{bmatrix} 0 & 0 \\ \varepsilon & 0 \end{bmatrix}, & & B_{s-1,s} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}, & & B_{s,s-1} = \begin{bmatrix} 0 & \mu \\ 0 & 0 \end{bmatrix} \end{array} \tag{1.57}$$

We recognise that several sub-matrices are actually identical. For the scheme to become apparent, we keep them separate because with the sub-matrices given, we can construct the Q-matrix: first combining the sub-matrices according to

$$Q = \begin{bmatrix} B_{00} & B_{01} & 0 & 0 & \dots & 0 \\ B_{10} & A_1 & A_0 & 0 & \dots & 0 \\ 0 & A_2 & A_1 & A_0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_2 & A_1 & B_{s-1,s} \\ 0 & \dots & 0 & 0 & B_{s,s-1} & B_{s,s} \end{bmatrix} \tag{1.58}$$

and than applying equation 1.53 to get the correct q_{ii} elements. For system size $s = 5$ (the example precisely shown in figure 1.20) we get

$$Q = \begin{bmatrix} -\varepsilon-\lambda & \varepsilon & \lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \varepsilon & -\varepsilon-\lambda & 0 & \lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu & -\mu-\lambda & 0 & \lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \varepsilon & -\varepsilon-\lambda & 0 & \lambda & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & -\mu-\lambda & 0 & \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \varepsilon & -\varepsilon-\lambda & 0 & \lambda & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu & -\mu-\lambda & 0 & \lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \varepsilon & -\varepsilon-\lambda & 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu & -\mu-\lambda & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \varepsilon & -\varepsilon-\lambda & 0 & \lambda \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu & -\mu & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \varepsilon & -\varepsilon \end{bmatrix} \tag{1.59}$$

being a typical Q-matrix with its characteristic diagonal shape. The only exception is the circle ε in the first row, the idle level, which causes the server to be equally likely available or away for the customers that happen to arrive when the system has been idle for some time. If the server would go on vacation faster in case no customer is there to serve, we would have to replace this rate by one over the time required to check if there is a customer or not. In case this time is zero, the state (0), the one without the $*$, and all connected transitions would vanish.

Having filled the Q-matrix we can calculate the state probabilities vector as described above for any given rates λ, μ, and ε, as shown for example in figure 1.21. Because we commonly solve such problems with the help of a computer for given rates only, as for example shown in algorithm 1.1, lets us call this the numeric approach. In theory the matrix equation can also be solved analytically with the according tools. However, the analytic equations generated by tools are often quite bulky, and challenging simplification is commonly out of the scope when tools are applied in the first place. Once coded in software every representation of the same equation yields the same results.

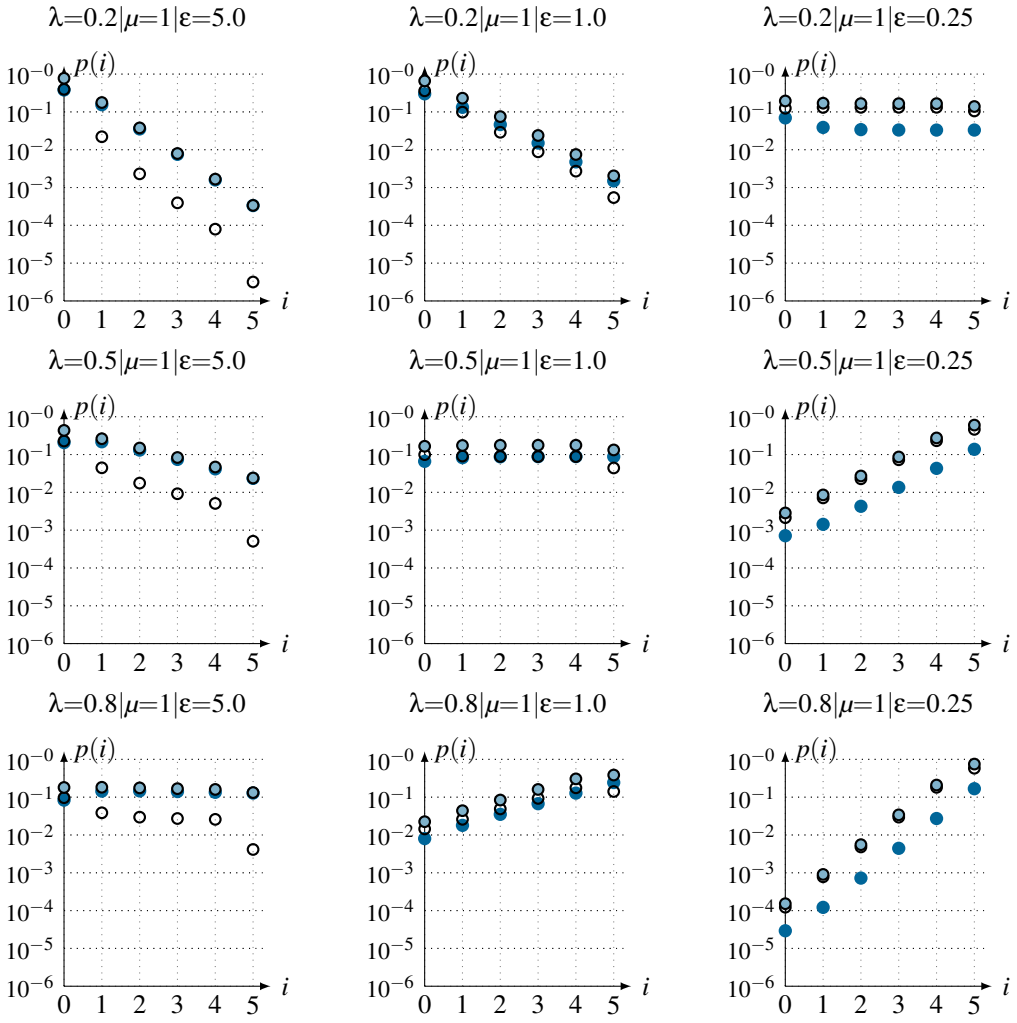


Figure 1.21: Vacation system state probabilities for different loads and vacation times
serving (dot), on vacation (circle), total (filled circle)

Alternatives and usability notes

Solving the equation system manually is no problem for systems with few states, and for systems with infinite states closed form solutions may be found if series equivalents can be found to mathematically formulate all transition rates without infinite sums. However, if the number of states is huge but not infinite, solving can become computationally intractable because of numeric issues with state probabilities becoming so small that they cannot be separated from numeric zero. In that case the solution must fail because zero probabilities may not occur for existing states. In that case we need to retreat to approximation. For example, an analytically solved infinite system of the same kind can be solved and truncated using the approximation proposed by Allen [28], or the behaviour of smaller systems may be extrapolated, if a clear system size related direction can be identified.

An alternative approach that should not be forgotten is finding a recursive definition for the state probabilities $p(i)$. Also that fits numeric calculation nicely:

- → select and assume one initial state probability, e.g., $\hat{p}_x = 1$
- → calculate all others based thereon one-by-one, $\hat{p}_i = f(\hat{p}_x)$
- → scale all $\hat{p}_i$ by a common factor $(\sum \hat{p}_i)^{-1}$ to get $\sum p_i = 1$.

Performing this analytically may be a hassle, because recursively inserting equations leads to increasingly lengthy expressions that only for special (simple) cases can be simplified conveniently.

Numerically, using some computer to do the repeated calculations, this approach features the option to choose an initially assumed probability high enough to not face numeric zeros in the course of the state-by-state calculations.

We use the Markov-chain approach to solve many, actually most of the systems discussed later on. Practically, we set up the matrix representation and let the freely available *Octave* software [29] solve the equation system repeatedly for changed rates, in order to get point-wise calculated curves. In this context we should also note that the mathematically undefined matrix division $p^T = b^{*T}/Q^{*T}$ as specified in the procedure shown in algorithm 1.1, actually performs an optimised *Gauss elimination*. This is more efficient than calculating the solution by matrix inversion $p = Q^{*(-1)} \cdot b^*$, as it is mathematically defined and would be performed if the division is not specified.

Algorithm 1.1 Vacation system state probabilities calculation procedure

```
function p=MM1s_pVac(lambda,mu,epsilon)

filename1=sprintf('MM1s_pVac_%3.3d_%3.3d_%3.3d_serving.dat',lambda*100,mu*100,epsilon*100);
filename2=sprintf('MM1s_pVac_%3.3d_%3.3d_%3.3d_vacation.dat',lambda*100,mu*100,epsilon*100);
filename3=sprintf('MM1s_pVac_%3.3d_%3.3d_%3.3d_total.dat',lambda*100,mu*100,epsilon*100);

systemsize=5; Qsize=2*(systemsize+1);
p=zeros(Qsize,1); Q=zeros(Qsize,Qsize); b=zeros(Qsize,1);

i=1; %fill the Q-matrix
Q(i,i+1)=epsilon; Q(i,i+2)=lambda; Q(i,i)=-sum(Q(i,:)); i++;
Q(i,i-1)=epsilon; Q(i,i+2)=lambda; Q(i,i)=-sum(Q(i,:)); i++;
while i<=Qsize-2
    Q(i,i-1)=mu;      Q(i,i+2)=lambda; Q(i,i)=-sum(Q(i,:)); i++;
    Q(i,i-1)=epsilon; Q(i,i+2)=lambda; Q(i,i)=-sum(Q(i,:)); i++;
endwhile
Q(i,i-1)=mu;      Q(i,i)=-sum(Q(i,:)); i++;
Q(i,i-1)=epsilon; Q(i,i)=-sum(Q(i,:));

Q(:,Qsize)=1; b(Qsize)=1; %insert sum(p)=1 condition
pt=transpose(b)/Q; %solve the equation system
p=transpose(pt) %get a vector again

p0=linspace(0,systemsize,systemsize+1); p0=transpose(p0);
p1=zeros(systemsize+1,1); p2=zeros(systemsize+1,1);
res1=fopen(filename1,"wt"); res2=fopen(filename2,"wt"); res3=fopen(filename3,"wt");
for i=1:Qsize/2
    p1(i)=p(2*i-1); p2(i)=p(2*i);
    fprintf(res1,'%3.0f %1.9f \n',p0(i),log10(p1(i)));
    fprintf(res2,'%3.0f %1.9f \n',p0(i),log10(p2(i)));
    fprintf(res3,'%3.0f %1.9f \n',p0(i),log10(p1(i)+p2(i)));
endfor
fclose(res1); fclose(res2); fclose(res3);
fprintf("Results saved in %s, %s, %s \n",myfilename1,myfilename2,myfilename3);

semilogy(p0,[p1 p2 p1+p2]); %plot the results
xlabel("number of customers i");
ylabel("state probability p(i)");
axis("labelxy");
thetitle=sprintf('M/M/1/s/vacation (%3.2f/%3.2f/%3.2f)',lambda,mu,epsilon);
title(thetitle);

endfunction
```

1.5 Distributions and statistical evaluation

Distributions describe the outcome of processes, meaning they are used to analytically approximate the true but in detail often imprecisely known process behaviour. In theory we want perfect models, in practice we commonly focus on how to identify the most simple distribution that models a process sufficiently well. First we introduce the most prominent distributions, and briefly repeat the basic theory and introduce the nomenclature used henceforth in this context. More distributions and their relevance to communication networks are presented and discussed in chapter 2. Next, the differences to *statistical evaluation of sample sets* are discussed and we introduce the calculation rules and schemes required to perform the calculation of *sample characteristics*, and show how these relate to the *moments of distributions*. Finally, we address the limits of statistical evaluation, and how the quality of statistical evaluation can be quantitatively expressed by *confidence intervals*.

1.5.1 Basic distributions

Distributions assign occurrence probabilities to the outcomes (events) of processes that commonly imply some randomness. They mathematically define the likelihood of any possible event. In engineering, distributions are commonly used where problems lead to differential equations whose solutions or initial conditions are somehow distributed. They enable us to derive the characteristics of technical processes without the need to evaluate it for every possible initial condition and possible sequence of events, which may be too excessive in practice (would not be scalable).

The simplest to understand distribution is the *uniform distribution* $U(a,b)$. All outcomes of a

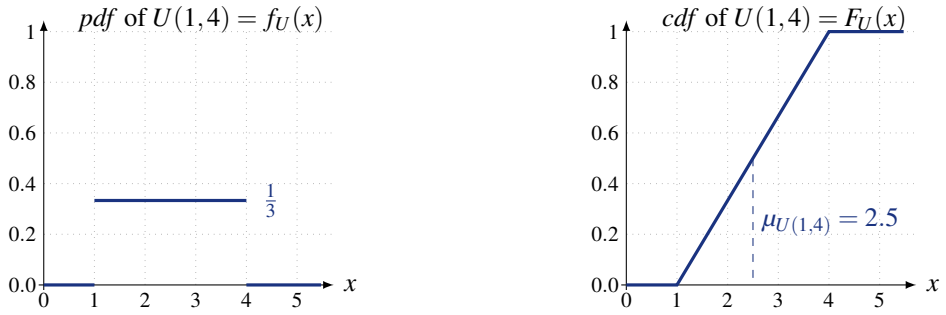


Figure 1.22: Uniform distribution $U(a,b)$: *pdf* and *cdf*

uniformly distributed random process are equally likely, and bound to the interval $(a\,..\,b)$, as shown in figure 1.22. Consequently is the probability of each possible event one over the width of the interval, because one of the possible outcomes needs to happen, $\sum p_i(x) = 1 \rightarrow \int_a^b f_U(x) = 1$.

$$pdf: \quad f_U(x) = \begin{cases} 0 & x \leq a \\ \dfrac{1}{b-a} & a < x \leq b \\ 0 & x > b \end{cases} \qquad cdf: \quad F_U(x) = \begin{cases} 0 & x \leq a \\ \dfrac{x-a}{b-a} & a < x \leq b \\ 1 & x > b \end{cases}$$

$$\text{mean:} \quad \mu_U = \frac{a+b}{2} \qquad \text{variance:} \quad \sigma_U = \frac{(b-a)^2}{12}$$

$$\text{matching:} \quad a = \mu_U - \sqrt{3\sigma_U} \qquad b = \mu_U + \sqrt{3\sigma_U}$$

Mathematically is the uniform distribution not very handsome because of its discontinuous *pdf*. However, there exist many very good pseudo-random number generators and therefore it is commonly used to compute samples of a certain distribution X by *inverse transform sampling*

$$X = F_X^{-1}\big(U(0,1)\big) \tag{1.60}$$

where $F_X^{-1}(u)$ is the inverse *cdf*, also called the *quantile function*. For somehow generated u values, uniformly distributed in $(0\,..\,1)$, the x-value for which $F_X(x) = u$ are distributed according to F_X. However, note that *pseudo-random number generators* may generate a true zero or one, which might cause numeric problems here. Luckily, the boundaries of $U(0,1)$ are here not relevant and without loss of generality we may adjust the boundaries to $0 < u < 1$ by skipping any u outside that region.

The Poisson process

To introduce the most convenient distributions we discuss the *Poisson process*. This is a very special process, which can be used to model continuous random processes with strictly positive valued, identically and independently distributed, outcomes. Note that the *Poisson distribution* is a discrete distribution, which results from the time continuous *Poisson process* shown in figure 1.23. The

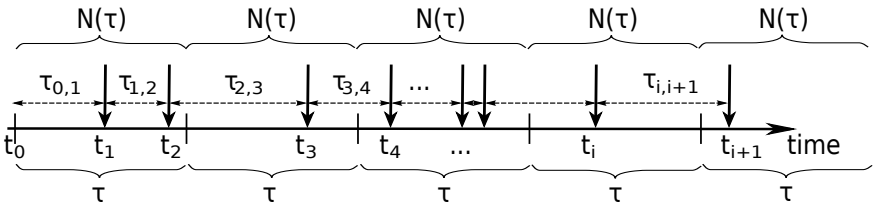


Figure 1.23: The Poisson process

Poisson process generates a *Poisson distributed* number of events $N(\tau)$ within each time interval τ

$$\mathrm{P}[N(\tau) = n] = \frac{(\lambda\tau)^n}{n!} e^{-\lambda\tau} \qquad \text{[Poisson distribution]} \tag{1.61}$$

where the events have *negative exponentially distributed* inter arrival times $\tau_{i,i+1} = t_{i+1} - t_i$

$$\mathrm{P}[T_{i,i+1} > \tau] = e^{-\lambda\tau} \qquad \text{[negative exponential distribution]} \tag{1.62}$$

and *uniformly distributed* occurrence times t_i within any time interval τ.

Therefore, the events are identically and independently distributed (i.i.d.), and the inter arrival times $\tau_{i,i+1}$ neither depend on the events' index i nor on an already passed time t

$$\mathrm{P}[T_{i,i+1} > t + \tau | T_{i,i+1} > t] = \mathrm{P}[T_{i,i+1} > \tau] = \mathrm{P}[N(\tau) = 0] = e^{-\lambda\tau} \tag{1.63}$$

because $\mathrm{P}[T_{i,i+1} > \tau]$ equals $\mathrm{P}[N(\tau) = 0]$, the probability that no events occur during τ. This special property expresses that the process is *memoryless*, and qualifies the Poisson process to be the principal time continuous Markovian process. The negative exponential distribution and its discrete counterpart, the geometric distribution, are the only memoryless distributions, and asymptotically result from superposing many *bursty* i.i.d. random variables with probability mass concentrated at zero, although these are by themselves not memoryless, see [30, 7.4] and *extreme value theory*.

The negative exponential distribution

Above we already encountered the *negative exponential distribution*, sketched in figure 1.24. Actually, it is the only *memoryless* continuous distribution and thus, it is commonly assumed when we refer to a Markovian distribution. Therefore, we use the index M to indicate it.

pmf: $f_M(x) = \lambda e^{-\lambda x}$ *cdf*: $F_M(x) = 1 - e^{-\lambda x}$

mean: $\mu_M = \dfrac{1}{\lambda}$

variance: $\sigma_M = \dfrac{1}{\lambda^2}$

matching: $\lambda = \dfrac{1}{\mu_M}$

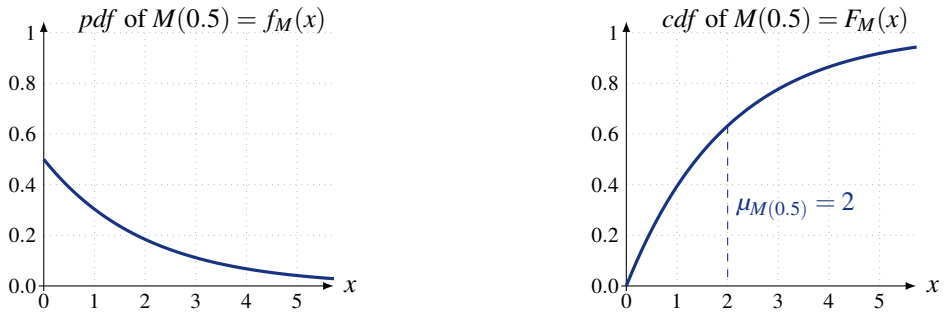


Figure 1.24: Negative exponential distribution $M(\lambda)$: *pdf* and *cdf*

As there is only one parameter to match with, the matching might be poor if the distribution type does not fit. However, being the only *memoryless* continuous distribution there is no alternative if we need to model a memoryless continuous random process.

Note that a continuous probability distribution is fully defined by its *cumulative distribution function* (*cdf*) or its *probability density function* (*pdf*). Latter is easier to interpret and shows the characteristics more clearly. The *cdf* is more handy for calculations and matching error evaluation. If we are particularly interested in a distributions tail, being the rarely occurring part to the far right, than we sometimes use the *complementary cdf* F_X^c (*ccdf*).

$$F_X^c = 1 - F_X$$

The geometric distribution

The discrete pendant to the negative exponential distribution is the geometric distribution shown in figure 1.25. It often occurs for discrete variables of an ergodic time continuous system driven by

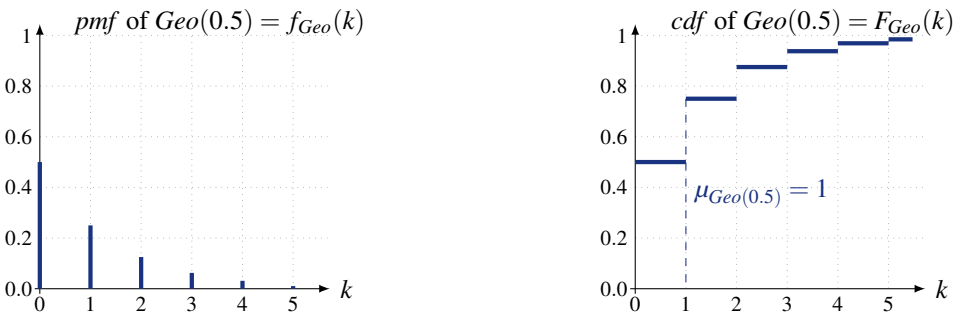


Figure 1.25: Geometric distribution $Geo(p)$: *pmf* and *cdf*

negative exponentially distributed arrivals. In the previous section 1.4 we derived for the M/M/1 queueing system that the probability for i customers in the system is $p(i) = \rho^i(1-\rho)$, and thus, it is geometrically distributed with $Geo(p = 1-\rho)$.

pmf: $f_{Geo}(k) = p\,(1-p)^k, \quad k \in \mathbb{N}^0$ *cdf*: $F_{Geo}(k) = 1-(1-p)^{k+1}, \quad k \in \mathbb{N}^0$

mean: $\mu_{Geo} = \dfrac{1-p}{p}$

variance: $\sigma_{Geo} = \dfrac{1-p}{p^2}$

matching: $p = \dfrac{\mu_{Geo}}{1+\mu_{Geo}}$

Here, $\mathbb{N}^0$ is the set of all natural numbers including zero. Again, we only have one parameter to match with, and thus, matching may be poor if the distribution type does not fit.

The geometric distribution is *memoryless*, alike its continuous analogue, the negative exponential distribution. This means that if an experiment is repeated until the first success, then the conditional probability distribution of the number of additional trials required until the first success, does not depend on how many failures have been observed. In easy terms: the die thrown or the coin tosses does not have a memory upon the failures that have occurred, the experiment proceeds independent of its past. Being the only *memoryless* discrete distribution, assuming it is a good guess if we need to model a memoryless discrete random process.

In the discrete domain, a probability distribution is fully defined by its *cumulative distribution function* (*cdf*) or its *probability mass function* (*pmf*). The discrete *cdf* is a discontinuous step function, though defined for all x; the *pmf* is a non-continuous comb function and only defined at discrete points. Mathematically not perfectly correct, we will refer to both by the acronyms *cdf* and *pdf*, respectively. When we henceforth talk about a discrete distribution's *pdf*, we notionally convert the discrete *pmf* into a continuous step-function and restrict evaluation to the positions just after the steps. Doing so we achieve identity.

$$\sum_{a}^{b} f_X(k) = \int_{a}^{b} f_X(x) \quad \forall_{a,b\in\mathbb{N}^0}$$

Formally, we use the *Dirac delta function* $\delta(k)$ to time continuously present a *pmf*, and its integral, the *unit step function* $u(k)$ to convert the discrete *cdf* into a function over continuous time.

1.5.2 Statistical values versus moments of distributions

The relations among statistical values and the moments and parameters of distribution functions are summarised in the following. First, we sketch the basic law that enables us to relate empiric statistics with analytic distributions.

The law of large numbers

$$\lim_{n\to\infty} \mathrm{P}\left[|\overline{X}_n - \mu| > \varepsilon\right] = 0 \quad \forall_{\varepsilon>0} \qquad \text{[weak law of large numbers]}$$

$$\mathrm{P}\left[\lim_{n\to\infty} \overline{X}_n = \mu\right] = 1 \qquad \text{[strong law of large numbers]}$$

The *strong law of large numbers* tells that for $n \to \infty$ repetitions of the same random experiment (X), the average over all collected results $x_{i=1\,..\,n}$, the *sample mean* $\overline{X}_n$, will converge to the actual mean $\mu = \mathrm{E}[X]$. The law of large numbers tells also that the *empirical probability* $\hat{p}_x$ of outcomes x in a series of independent and identically distributed experiments will converge to the result probabilities p_x. This interpretation promises that we can approximate the *pdf* over x, and thereby identify a distribution f_X that approximately describes the random experiment analytically.

For a sequence of random experiment outcomes $\{x_1, x_2, \ldots, x_n\}$, the strong law of large numbers assures us that the *empirical sample mean* $\hat{\mu} = \overline{X}_n$ will almost surely converge to the *theoretic mean* μ. Consequently, to perfect calculate the expectation value $\mathrm{E}[X]$ an infinite sample set would be required, which is not possible in practice. The statistical uncertainty intrinsic to the evaluation of finite sample sets is expressed by *confidence intervals* $[\mu-\zeta \,..\, \mu+\zeta]$, discussed in section 1.5.3. Only for sufficiently large sample sets we may assume that the difference between the calculated average and the true expectation value is negligible, as it is stated by the *weak law of large numbers*. Vice versa, we conclude that a single experiment, tells us nearly nothing about the random process, only that the found sample is in the sample space Ω_X.

Next we briefly summarise some concepts of statistics useful for the distributions and sample sets we commonly encounter. For in depth information please refer to the rich literature on statistics and stochastic, for example [28, 31–33], and note that hereinafter the term *mean* will most commonly refer to the true $\mathrm{E}[X] = \mu$, and *average* to the sample mean $\overline{X}_n = \mathrm{E}[X_n]$.

The mean and the average

$$\begin{aligned}
&\text{mean of continuous distribution} && \mathrm{E}[X] = \int_{-\infty}^{\infty} \mathrm{P}[X \geq x]\,\mathrm{d}x = \int_{-\infty}^{\infty} x f_X(x)\,\mathrm{d}x \overset{f_X(x<0)=0}{=} \int_{0}^{\infty} (1-F(x))\,\mathrm{d}x \\
&\text{mean of discrete distribution} && \mathrm{E}[X] = \sum_{x=-\infty}^{\infty} \mathrm{P}[X \geq x] = \sum_{x=0}^{\infty} x\,\mathrm{P}[X=x] \overset{\mathrm{P}[X<0]=0}{=} \sum_{x=0}^{\infty} x p_x \\
&\text{average of sample set} && \mathrm{E}[X_n] = \overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} x_i \in X_n
\end{aligned} \tag{1.64}$$

In all cases we use the expectation operator $\mathrm{E}[\,]$, but note that quite different operations define it for the different cases. If the sample set $X_n = \{x_i\}_n$ is the outcome of an analytic defined process X, meaning defined by its *cdf* $F_X(x)$ or *pdf* $f_X(x)$, than we get: $\mu_X = \lim_{n\to\infty} \mathrm{E}[X_n] = \mathrm{E}[X]$.

Note that the further right parts of equation 1.64 are applicable for positive valued normalised probability distributions only, when $F_X(x{<}0){=}0$, $f_X(x{<}0){=}0$, and also $F_X(\infty){=}1$, and $\int f_X(x){=}1$.

The *expectation operator* $\mathrm{E}[\,]$ is linear in the sense that

$$\begin{aligned}
&\text{we can extract factors} && \mathrm{E}[aX] = a\,\mathrm{E}[X] \\
&\text{we can extract constants} && \mathrm{E}[X{+}c] = \mathrm{E}[X] + c \\
&\text{we can sum expectations} && \mathrm{E}[X{+}Y] = \mathrm{E}[X] + \mathrm{E}[Y] \\
&\text{and we thus can simplify} && \mathrm{E}[aX{-}bY{+}c] = a\,\mathrm{E}[X] - b\,\mathrm{E}[Y] + c
\end{aligned} \tag{1.65}$$

and note that the summing of expectations is valid even if X is not statistically independent of Y. However, the expectation operator is in general not multiplicative $\mathrm{E}[XY] \neq \mathrm{E}[X]\,\mathrm{E}[Y]$.

To calculate the *joint expectation* $\mathrm{E}[XY]$ we need the *joint probability density function* $f_{X,Y}(x,y)$.

$$\begin{aligned}
&\text{the joint expectation} && \mathrm{E}[XY] = \iint xy f(x,y)\,\mathrm{d}x\,\mathrm{d}y \\
&\text{and the covariance} && Cov(X,Y) = \mathrm{E}[XY] - \mathrm{E}[X]\,\mathrm{E}[Y]
\end{aligned} \tag{1.66}$$

If $Cov(X,Y){=}0$ than X and Y are said to be *uncorrelated*, and $f_{X,Y}(x,y) = f_X(x) f_Y(y)$, using the marginal *pdf*s. For *independent* random variables this is generally the case, and we can calculate the joint expectation from the individual expectations: $\mathrm{E}[XY] = \mathrm{E}[X]\,\mathrm{E}[Y]$. However, being uncorrelated is not a sufficient condition for being independent.

The *conditional pdf* $f_X(x|Y{=}y)$ and the *conditional expectation* $\mathrm{E}[X|Y{=}y]$ are calculated as

$$\begin{aligned}
&\text{the conditional } pdf && f_X(x|Y{=}y) = \frac{f_{X,Y}(x,y)}{f_Y(y)} = f_Y(y|X{=}x)\frac{f_X(x)}{f_Y(y)} \quad \text{and} \\
&\text{the conditional expectation} && \mathrm{E}[X|Y{=}y] = \sum x\,\mathrm{P}[X{=}x|Y{=}y] = \sum \frac{x\,\mathrm{P}[X{=}x, Y{=}y]}{\mathrm{P}[Y{=}y]}
\end{aligned} \tag{1.67}$$

and we recognise that they both depend on the joint probability, $f_{X,Y}(x,y)$ and $\mathrm{P}[X{=}x, Y{=}y]$, respectively. If and only if the conditional *pdf* or expectation equal the non-conditional, than the two random variables X, Y are *independent*.

Bayes' law relates conditional probabilities by

$$\mathrm{P}[X|Y] = \frac{\mathrm{P}[Y|X]\,\mathrm{P}[X]}{\mathrm{P}[Y]} \tag{1.68}$$

and this is often essentially helpful. Also note that joint distributions and probabilities are independent of the joining direction: $f_{X,Y}(x,y) \equiv f_{Y,X}(y,x)$, and more evidently $\mathrm{P}[X{=}x, Y{=}y] \equiv \mathrm{P}[Y{=}y, X{=}x]$.

The variance and the standard deviation

$$Var[X] = \mathrm{E}\left[(X-\mu)^2\right] \tag{1.69}$$
$$= Cov(X,X) = \mathrm{E}\left[X^2\right] - \mathrm{E}[X]^2 \tag{1.70}$$

The variance indicates how far the outcomes of a random process are spread out, being a metric for the mean divergence from the mean. The variance of a real-valued random variable is its second central moment, and thus the first characteristic that can be used to distinguish between different probability distributions. It can be calculated either directly via the squared divergence of each sample from the true mean $(X-\mu)^2$ (equation 1.69), or using the properties of the covariance, meaning $Cov(X,X)$, by subtracting the squared mean $\mathrm{E}[X]^2$ from the mean of squares $\mathrm{E}\left[X^2\right]$ (equation 1.70).

Depending on the type of random variable, we can calculate the variance by

variance of continuous distribution
$$Var[X] = \int (x-\mu)^2 f_X(x)\,\mathrm{d}x = 2\int_0^\infty x F_X^c(x)\,\mathrm{d}x - \left(\int_0^\infty F_X^c(x)\,\mathrm{d}x\right)^2$$

variance of discrete distribution
$$Var[X] = \sum_{i=1}^{n} p_i\,(x_i-\mu)^2$$

variance of sample set
$$Var[X_n] = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \hat{\mu}^2 = \frac{\left(\sum_{i=1}^n x_i^2\right)}{n} - \frac{\left(\sum_{i=1}^n x_i\right)^2}{n^2} \tag{1.71}$$

The *variance* $Var[\,]$ is non-negative, zero if the samples are not distributed, and may be infinite for very heavy tailed distributions. However, we can

extract factors
$$Var[aX] = a^2 Var[X]$$

ignore constants
$$Var[X+c] = Var[X]$$

solve weighted sums
$$Var[aX+bY] = a^2 \mathrm{Var}[X] + b^2 Var[Y] + 2ab\,Cov(X,Y) \tag{1.72}$$

do multiple summation
$$Var\left[\sum_{i=1}^{n} X_i\right] = \sum_{i,j=1}^{n} Cov(X_i,X_j) = \sum_{i=1}^{n} Var[X_i] + \sum_{i\neq j} Cov(X_i,X_j)$$

and consequently is the variance of a finite weighted sum of *uncorrelated* random variables, for which $Cov(X,Y) = 0$, equal to the square weighted sum of their variances.

$$Var\left[\sum_{i=1}^{n} a_i X_i\right] = \sum_{i=1}^{n} a_i^2 Var[X_i] \tag{1.73}$$

If two distributions are *independent*, than we can directly calculate the joint variance of their product.

$$Var[XY] = \mathrm{E}[X]^2\,Var[Y] + \mathrm{E}[Y]^2\,Var[X] + Var[X]Var[Y] \tag{1.74}$$

The variance of a group of random outcomes comprising X is equal to the mean of the variances of equally sized subgroups plus the variance of the means of the subgroups. This property is known as *variance decomposition* or *the law of total variance* and plays an important role in analysis. So if X and Y are two random variables and the variance of X exists, then $Var[X]$ can be calculated by

$$Var[X] = \mathrm{E}[Var[X|Z]] + Var\left[\mathrm{E}[X|Z]\right] \tag{1.75}$$

where $\mathrm{E}[X|Z]$ and $Var[X|Z]$ are *conditional expectation* and *conditional variance* of X given Z, respectively. More generally states the *law of total covariance* that we can calculate the $Cov(X,Y)$ by

$$Cov(X,Y) = \mathrm{E}[Cov(X,Y|Z)] + Cov(\mathrm{E}[X|Z], \mathrm{E}[Y|Z]) \tag{1.76}$$

where again Z states the conditions by which the subsets are created.

Because the variance is a square metric, also its unit is squared. In practice it is often more intuitive to specify the *standard deviation* σ_X (equation 1.77), also called *root mean square deviation*, expressing its origin. Being the square root of the variance, it has the same unit as the mean. The *sample deviation* $\varsigma[X_n]$ (equation 1.78), being the *root average square deviation* from the *sample average* $\hat{\mu}=\frac{1}{n}\sum x_i$, is defined as the square root of the *sample variance* ς^2. Only similar in its unit is the *average absolute deviation* $\partial_a[X]$ (equation 1.79).

standard deviation
$$\sigma_X = \sqrt{Var[X]} = \sqrt{\mathrm{E}[X^2] - \mathrm{E}[X]^2} \tag{1.77}$$

sample deviation
$$\varsigma[X_n] = \sqrt{\varsigma[X_n]^2} = \sqrt{\tfrac{1}{n-1}\sum (x_i - \hat{\mu})^2} \tag{1.78}$$

average absolute deviation
$$\partial_a[X] = \mathrm{E}[|X-\mu|] \leq \sigma_X \tag{1.79}$$

Higher moments and characteristics

Commonly we are satisfied if the first and the second moment match, and the consideration of higher moments is rarely required. However, if we have a sample set and want to characterise the underlying distribution, than the higher moments can tell us some details about the distribution, which cannot be derived from the first two moments alone. Therefore, we introduce them briefly.

Several times we mentioned that mean and variance are moments of the distribution. Actually is the mean the first *raw* moment, and the variance is the second *central* moment. Raw moments result from

$$\mu^{(n)} = \int_{-\infty}^{\infty} x^n f(x)\,\mathrm{d}x,$$

while central moments are calculates in respect to the mean value by

$$\mu^{(n)} = \int_{-\infty}^{\infty} (x-\mu)^n f(x)\,\mathrm{d}x,$$

and evidently is $\mu^{(0)} = \int f(x)\,\mathrm{d}x = 1$, because the area below the *pdf* $f(x)$ is always one. For higher moments we prefer to use the *standardized* moments, more precisely called *normalized n^{th} central moment*

$$\mu^{(n)} = \frac{1}{\sigma^n}\int_{-\infty}^{\infty} (x-\mu)^n f(x)\,\mathrm{d}x,$$

because these are dimensionless quantities that characterise the distribution independent of any linear scale change. In other words, they tell us something about the distributions shape.

The normalized third central moment is called *skewness* γ, and is a measure of the lopsidedness of the distribution. For a symmetric distribution is $\gamma = 0$, if it exists. A distribution skewed to the right, meaning that its tail is heavier on the right side, has a positive skewness. Vice versa has a left skewed distribution a negative skewness. If two distributions are *independnet*, than the skewness is additive: $\gamma(X+Y) = \gamma(X)+\gamma(Y)$. The fourth central moment indicates whether the distribution is tall and skinny or short and compact, compared to the normal distribution with same variance.

This moment minus 3 is called *kurtosis* κ. If a distribution has a peak at the mean and long tails, the fourth moment will be high and the kurtosis positive. Conversely, bounded distributions, having short tails, tend to have low kurtosis. Independent of how it is defined, if the n^{th} moment exists, all lower moments exist, and if the n^{th} moment does not exists, than no higher moment exists. The larger the higher moments are, the harder it is to estimate the characteristics from a finite sample set.

The moment generating function

$$M_X(t) = \mathrm{E}\left[e^{-tX}\right] = \int_{-\infty}^{\infty} e^{-tx} f_X(x)\,\mathrm{d}x, \quad t \in \mathbb{R} \tag{1.80}$$

offers an alternative specification for a random variables probability distribution in addition to the *pdf* or the *cdf*. However, the moment-generating function does not always exist, unlike the characteristic function sketched next.

The prime reason for using this function is that by it we can find all moments of a distribution. Using the series expansion $e^{tX} = \sum_0^\infty \frac{t^i X^i}{i!}$ we get for n being the rank of the highest existing moment

$$M_X(-t) = \mathrm{E}\left[e^{tX}\right] = 1 + t\mu^{(1)} + \frac{t^2\mu^{(2)}}{2!} + \cdots + \frac{t^n\mu^{(n)}}{n!} = \sum_0^n \frac{t^i\mu^{(i)}}{i!}$$

where $\mu^{(i)}$ is the i^{th} moment, which can be calculated by i-times differentiating $\mathrm{d}^i M_X(-t)/\,\mathrm{d}t^i$ and setting t=0 in the result.

For the weighted sum of *independent* random variables $X = \sum a_i X_i$, for which the *pdf* $f_X(x)$ is the consecutive *convolution* of the *pdf*s $f_{X_i}(x)$ of each X_i, we get for the joint moments generating function.

$$M_X(t) = \prod M_{X_i}(a_i t)$$

The characteristic function

$$\varphi_X(t) = \mathrm{E}\left[e^{itX}\right] = \int_{-\infty}^{\infty} e^{itX}\mathrm{d}F_X(x) = \int_{-\infty}^{\infty} e^{itx} f_X(x)\,\mathrm{d}x, \quad i = \sqrt{-1},\ t \in \mathbb{R} \tag{1.81}$$

is defined as the expected value of e^{itX}, where i is the imaginary unit, and $t \in \mathbb{R}$ is the argument of the characteristic function. It also completely determines the behaviour and properties of any real-valued probability distribution. The $\mathrm{d}F_X(x)$ integral is of the Riemann-Stieltjes kind, and thus, for a scalar random variable X with existing *pdf*, the characteristic function is the inverse Fourier transform of it, such that the rightmost part in equation equation 1.81 is valid, and we can derive the *pdf* therefrom. The properties of characteristic functions include

it always exists	for real valued random variables
is uniformly continuous	on the entire space
is bounded	$\varphi_X(t) \le 1$
it defines all moments	if $\mu^{(k)}$ exists: $\mu^{(k)} = \mathrm{E}\left[X^k\right] = (-i)^k \varphi_X^{(k)}(0) = (-i)^k \left.\frac{\mathrm{d}^k \varphi_X(t)}{\mathrm{d}t^k}\right\vert_{t=0}$
is invertible	if φ_X is integrable: $f_X(x) = \frac{\mathrm{d}F_X(x)}{\mathrm{d}x} = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-itx}\varphi_X(t)\,\mathrm{d}t$

and again is the characteristic function of the weighted sum of independent random variables the product of the scaled individual characteristic functions $\varphi_{X_i}(a_i t)$

$$\varphi_X(t) = \prod \varphi_{X_i}(a_i t), \qquad \text{if } X = \sum a_i X_i,\ Cov(X_i, X_j) = 0\ \forall_{i \neq j}$$

and for the *sample mean* $\overline{X}_n = \frac{1}{n} \sum_{i=1}^{n} x_i$ we get as its characteristic function.

$$\varphi_{\overline{X}_n}(t) = \left(\varphi_X(\tfrac{1}{n} t)\right)^n$$

We note that the constants appearing in equation 1.81 differ from the usual used with the Fourier transform. Consequently, some authors use $\varphi_X(t) = \mathrm{E}\left[e^{-2\pi i t X}\right]$, where t is changed to $-2\pi t$. Other, similarly altered notations and accordingly adjusted properties may be encountered in the literature.

1.5.3 Confidence intervals

Statistical values are in general calculated from many individual samples forming a *sample set*, called *trace* if time instances are bundled with the individual samples. Evaluating independent repetitions of the same random experiments yields different results for each, as for example shown in figure 1.26 for the sample mean $\mathrm{E}[X_n]$ of small sample sets ($X_{[10]}$) drawn from the uniformly distributed sample

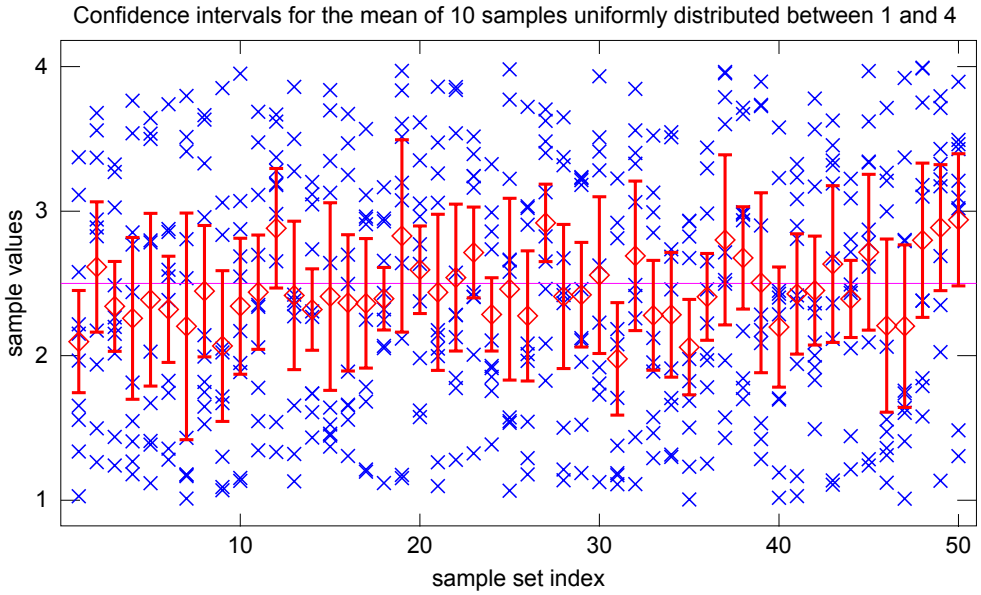


Figure 1.26: 95% confidence intervals of repeated experiments

space Ω_X of $U(1,4)$. The x-axis refers to the experiment index, on the y-axis the × are the samples, the ◇ the sample means, and the bars I depict the calculated 95% confidence intervals. Experiments number 1, 27, 31 and 35 happen to show confidence intervals that actually do not enclose the true mean $\mathrm{E}[X]$=2.5. That 4 out of 50 intervals do not enclose the true $\mathrm{E}[X]$ is pure chance, in average it should be 5% only.

Confidence intervals need to be shown to express the systematic uncertainty inherent to finite sample sets. Even though the statistical estimates approach the characterising parameters of the true distribution with increasing sample size, in respect to the *law of large numbers* (see 1.5.2), we cannot exclude the probability that an outcome, in particular the current sample set used for statistical evaluation, poorly represents the whole sample space Ω_X. Therefore, confidence intervals $[(\overline{X}_n - \zeta_-), (\overline{X}_n + \zeta_+)]$ are commonly reported in tables or graphs along with the point estimates $\overline{X}_n$

of calculated parameters in order to provide an indication for the systemic reliability of the calculated estimates.

$$\mathrm{P}\big[\overline{X}_n - \zeta_- < \mathrm{E}[X] < \overline{X}_n + \zeta_+\big] = L^C$$

We note that these are easily misinterpreted: to be mathematically correct we may not say that the true value $\mathrm{E}[X]$ is with the probability L^C within a calculated confidence interval. We can only say that with the chosen *confidence level* L^C, commonly 95%, a calculated interval will enclose the true value $\mathrm{E}[X]$. The difference is subtle, but note that a particular confidence interval may be offset by a biased $\overline{X}_n$, such that the true value $\mathrm{E}[X]$ is likely not enclosed. Thus, for any particular confidence interval I^C is $\mathrm{P}\big[\mathrm{E}[X] \in I^C\big] \neq L^C$. Actually, we cannot say anything about the $\mathrm{P}\big[\mathrm{E}[X] \in I^C\big]$, because we have no information on how good the available samples $x_i \in X_n$ represent the sample space Ω_X.

The correct explanation for L^C is: if we repeat an experiment many times, than a proportion approximately equal to the chosen confidence level L^C of the different confidence intervals I_i^C, which we get for the individual experiments, will actually enclose the true value $\mathrm{E}[X]$

$$\mathrm{P}\big[(I_i^C \,|\, \mathrm{E}[X] \in I_i^C)\big] = L^C \quad i = 1,2,3,\ldots,$$

where I_i^C is one out of many independently calculated confidence intervals, and L^C is the chosen confidence level.

$$\lim_{N\to\infty}\left(\frac{1}{N}\sum_{i=1}^{N}\xi_i\right) = L^C, \quad \xi_i = \begin{cases} 1 & \text{if } \mathrm{E}[X] \in I_i^C \\ 0 & \text{if } \mathrm{E}[X] \notin I_i^C \end{cases}$$

Confidence interval calculation

While the theory of confidence intervals, more generally on interval estimates, is apparent, the calculation of confidence intervals is a little ambiguous and the topic commonly appears in the far end of text books. The key problems are:

(a) independence among samples, $Cov(X_i, X_{i+k})=0\ \forall_k$, is presumed but cannot be guaranteed,
(b) direct calculation requires the true $Var[X]=\sigma^2$ but we only know the sample variance ς^2,
(c) engineering science is anxious about correctness and the aspired high confidence in calculations seems toppled by low confidence levels L^C or wide confidence intervals I^C.

These problems are systemic and cannot be resolved. Problems (a) and (b) result from the core assumption that the sample estimates $\overline{X}_n$ are *normal* distributed around the true estimate $\mu = \mathrm{E}[X]$, by applying the central limit theorem on the contribution of each sample on the estimate. This forces independent and identically distributed samples. Without assuming a distribution, a direct calculation is not possible. Shortly, in 1.5.4, we sketch methods to improve the calculation in cases where the direct approach yields too poor results.

The latter problem (c) results from the fiction that statistical results could be 100% correct. Actually, even if the calculations are 100% correct, the correctness of calculated expectations is always limited by the available data. How poor the available data is, we express by showing confidence intervals. Thus, wide confidence intervals improve the quality of calculations, because they add valuable information to the corresponding point estimates.

For means of comparability we calculate confidence intervals with the common confidence level $L^C = 95\%$, if not stated differently. In any case, for a reliable statistical evaluation the sample size n needs to be sufficiently large, and for efficiency not larger than required.

The direct calculation [34] is based on Chebychev's equation

$$\mathrm{P}\big[|\overline{X}_n - \mu| \geq \varepsilon\big] \leq \frac{\sigma^2/n}{\varepsilon^2}, \tag{1.82}$$

which can be rewritten as

$$\mathrm{P}\big[-\varepsilon < \overline{X}_n - \mu < \varepsilon\big] = \mathrm{P}\big[|\overline{X}_n - \mu| < \varepsilon\big] \geq 1 - \frac{\sigma^2}{n\varepsilon^2} \tag{1.83}$$

to show its relevance for confidence intervals. If we assume that the samples $x_i \in \Omega_X$ are *normal* distributed with $N(\mu,\sigma^2)$, than we know that the sample means $\overline{X}_n$ of different sample sets holding the same number of samples n each, are *normal* distributed with $N(\mu, \frac{\sigma^2}{n})$. If we substitute $Z = \frac{\overline{X}_n - \mu_X}{\sigma/\sqrt{n}}$, which is distributed with $N(0,1)$, we get

$$\mathrm{P}[|Z| < z] = \mathrm{P}\left[\frac{|\overline{X}_n - \mu|}{\sigma/\sqrt{n}} < z\right] = \mathrm{P}\left[|\overline{X}_n - \mu| < \frac{z\sigma}{\sqrt{n}}\right] = \mathrm{P}\left[\mu - \frac{z\sigma}{\sqrt{n}} < \overline{X}_n < \mu + \frac{z\sigma}{\sqrt{n}}\right] = L^C$$

and we get for the two half widths that

$$\mathrm{P}[Z < -z] = \mathrm{P}[Z > z] = \frac{1 - L^C}{2}.$$

Consequently, we can calculate the bounds of the intended confidence interval by selecting the z that fits to the chosen L^C (table 1.2), and inserting it in

$$\left(\overline{X}_n - \frac{z\sigma}{\sqrt{n}}, \overline{X}_n + \frac{z\sigma}{\sqrt{n}}\right). \tag{1.84}$$

Table 1.2: z values for $X_{n>100}$ holding normal distributed samples x_i [33–35]

L^C	80%	90%	**95%**	99%	99.5%	99.9%	99.99%	99.999%	99.9999%
z	1.282	1.645	**1.960**	2.576	2.807	3.291	3.891	4.417	4.892

The half-width of confidence intervals, being $z\sigma/\sqrt{n}$, is determined by the number of samples within the available sample set n, the standard deviation σ, and z, which is determined by the chosen confidence level L^C. To halve the width of the confidence interval for a given L^C we either need a fourfold sample size n, or need to reduce the variance by 50% [34]. Thus, variance reduction techniques are more scalable, but being unpredictably effective, we do not consider them henceforth.

We note that for the confidence interval calculation in equation 1.84 the true standard deviation σ is used, which cannot be derived from the sample set X_n. In practice we use the unbiased sample variance $\varsigma^2 = \frac{1}{n-1}\sum(x_i - \overline{X}_n)^2$ and hope that the introduced error is $\ll 1 - L^C$. It is apparent, that for high L^C this assumption is hardly justified. Only for large n the *law of large numbers*, and for independent and identically distributed samples the *central limit theorem*, assures that ς approximates σ.

For small n we have to use $Z = \frac{\overline{X}_n - \mu_X}{\varsigma/\sqrt{n}}$, and utilize the property that this random variable is known to be *Student t*-distributed with $\nu = n-1$ degrees of freedom. For large n this distribution approaches the normal distribution, and for trustworthy results $n \geq 10$ is strongly recommended. Tables for the *Student t-distribution* are available, an excerpt is shown in table 1.3. This replaces table 1.2 whenever we need to calculate confidence intervals from small sample sets composed of samples that are potentially not independent and identically distributed.

Confidence interval calculation from measurement and simulation results

Sample sets collected during an entire measurement or simulation run are in case of traces a sequence of transient phases and steady phases. Good steady state estimates can only be calculated from samples collected during steady phases, and therefore, samples collected during transient phases should be sorted out. Either this is performed a priori during the sample collection, or we need to perform this a posteriori on the sample set. Latter can be quite challenging, and, it may lead to overly smoothed sample sets if excessively applied. Typically we only need to sort out the transient phase prior reaching the steady state and the typically rather short fade out phase in the end of a trace.

Table 1.3: z values based on Student's t-distribution [33, 35, 36]

$\nu \backslash L^C$	80%	90%	**95%**	99%	99.5%	99.9%	99.99%	99.999%
∞	1.282	1.645	**1.960**	2.576	2.807	3.291	3.891	4.417
100	1.290	1.660	**1.984**	2.626	2.871	3.390	4.053	4.654
80	1.292	1.664	**1.990**	2.639	2.887	3.416	4.096	4.717
60	1.296	1.671	**2.000**	2.660	2.915	3.460	4.169	4.825
40	1.303	1.684	**2.021**	2.704	2.971	3.551	4.321	5.053
20	1.325	1.725	**2.086**	2.845	3.153	3.850	4.837	5.854
10	1.372	1.812	**2.228**	3.169	3.581	4.587	6.211	8.150
8	1.397	1.860	**2.306**	3.355	3.833	5.041	7.120	9.783
6	1.440	1.943	**2.447**	3.707	4.317	5.959	9.082	13.56
4	1.533	2.132	**2.776**	4.604	5.598	8.610	15.54	27.77
2	1.886	2.920	**4.303**	9.925	14.09	31.60	99.99	316.2

The common approach is to split the entire trace into several intervals, each containing the same number of samples n_i=n, except the last, for which $n_i \leq n$ is sufficient. To discard systematic transient phases we commonly calculating estimates not considering the first and the last interval. Thus, we need to choose n such that we can be sure that all transient start-up effects fade within one interval, assuring that after the first interval the steady state is reached. In figure 1.27 we actually show generated samples and the windowed mean as the example trace. At the begin there is a

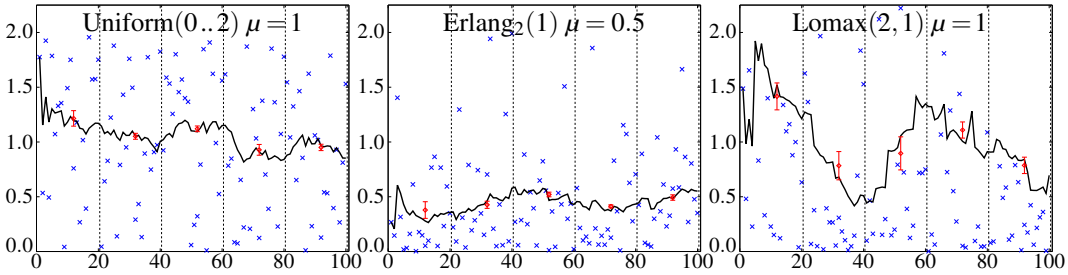


Figure 1.27: Example traces (—) generated from Uniform/Erlang/Lomax distributed samples (×) showing batch means (◇) and variances (I) of the windowed mean calculation (—)

transient phase but due to the very short averaging window of 20 samples the mean does not stabilise much, in particular not when the variance is high. However, assuming independent and identically distributed mean values per interval (*batch*) allows use to directly calculate the total *point estimate* and its *confidence interval*

batch means
$$\overline{X}_i = \frac{1}{n}\sum_{j=1}^{n} \mathbf{S}\left[(i-1)n+j\right]$$

total mean
$$\overline{X}_N = \frac{1}{\lfloor N/n \rfloor - 1}\sum_{i=2}^{\lfloor N/n \rfloor} \overline{X}_i$$

variance of batch means
$$\varsigma^2_{\overline{X}_i} = \frac{1}{\lfloor N/n \rfloor - 2}\sum_{i=2}^{\lfloor N/n \rfloor} \left(\overline{X}_i - \overline{X}_N\right)^2$$

confidence interval
$$\left(\overline{X}_N - \frac{z\,\varsigma_{\overline{X}_i}}{\sqrt{\lfloor N/n \rfloor}}, \overline{X}_N + \frac{z\,\varsigma_{\overline{X}_i}}{\sqrt{\lfloor N/n \rfloor}}\right)$$

where N is the size of the entire trace and $\lfloor N/n \rfloor$ is the number of intervals except the last, and thus is $\lfloor N/n \rfloor - 1$ the number of batches. To calculate confidence intervals (using the equation above) we insert for z the figures given in table 1.2, if we may assume normal distributed batch means,

meaning independent and identically distributed samples throughout all considered intervals. If we cannot assume that the samples of all batches are independent and identically distributed, we insert the figures given in table 1.3 for the according number of batch means we average over, only assuming that the batch means are independent and identically distributed. Latter precondition is fulfilled if any autocorrelation fades to negligible well within n, the size of the batches. The common recommendation is $n \geq 5 \cdot \ell$, where $\ell = \min\left(k | Cov(X_i, X_{i+j}) \leq \varepsilon, \forall_{j \geq k}\right)$ [34], in order to be sure that each batch contains a sufficient number of uncorrelated samples.

Generally, we need sufficiently large batch sizes n to calculate good estimates. Here we might reduce n without loosing precision because what actually counts is the total number of samples $n \cdot \lfloor N/n \rfloor$ that we actually use to calculate $\overline{X}_N$. The more batches, the better can the batch mean variance be estimated and thereby the confidence interval calculation improved. Consequently, the optimal batch size n is sufficiently large to let autocorrelations fade, and small enough to grant a sufficient number of batches for the confidence interval calculation. For batches holding by design independent and identically distributed samples we can presume that the distribution of the batch means $\overline{X}_i$ is approximately normal distributed as a consequence of the *central limit theorem*. In that case equation equation 1.84 and table 1.2 may be used to calculate correct confidence intervals, as we did to create figure 1.26.

To better handle cases with potential transient phases anywhere within a trace, we may exclude all intervals that do not conform with the majority of intervals. To minimise the number of thereby lost samples we should optimise n such that the number of samples contained in not considered *outlier* intervals is minimised. Anyhow, we have to be sure that we may exclude any *outlier* intervals determined because else we unduly refurbish the result.

1.5.4 Statistical inference techniques

Statistical inference is used to draw conclusions from finite sample sets challenged by inevitable random variation. The target is to identify observation errors and to mitigate sampling variation. Based on the derived characteristics of the sample set, its quality is validated, and this provides us a statistical proposition on how useful the available sample set likely is.

A key problem to be solved is the finiteness of the available sample set. Assuming that samples are independent and identically distributed, we can create alternate sample sets from the available samples. The *jackknife* and the *bootstrap* scheme are popular *resampling methods* used in statistical analysis, whereas *Monte Carlo sampling* is popular with statistical software tools. Besides the theoretical differences in their mathematical fundamentals, a practical difference is that the bootstrap produces different results when repeated on the same data, whereas the jackknife yields exactly the same result each time.

Jackknifing

is used in statistical inference to estimate the estimation bias and standard error, when only one random sample set of observations is available. The method of subsequent sample-sets outlined in section 1.5.3 is a very simple version of jackknifing. Independent on how ingenious it is realised, the core idea of the jackknife variance estimator relies in systematic recomputing of the estimate while leaving out different observations. Systematic, because we assures that in the end all available samples have been used equally often. For example once only, as in 1.5.3 or the here sketched algorithm 1.2. The more the subsets overlap the more jackknife estimates we get. To consider all possible permutations will in general be too excessive.

Note that $\frac{N}{M}$ needs to be integer, meaning that the number of jackknife samples M that we create needs to be a divisor of the full sample set's size N, and the number of samples within each jackknife sample $\overline{J}_i$ is a fraction of the total number of samples available. Consequently, this method is best applicable for huge original sample sets $\mathbf{S}_N$. Using the jackknife estimates we get for the sub-sets $\overline{J}_i$

Algorithm 1.2 jackknife(sample set $\mathbf{S}_N$, confidence level L^C, M)

sorted set $\mathbf{J} = \{\}$
for $i = 1 \,..\, M$ **do**
  $m = 0$
  for $j = 1 \,..\, \frac{N}{M}$ **do**
    $x = i + (j-1)M$
    $m = m + \mathbf{S}[x]$
  end for
  $\overline{J}_i = m/\frac{N}{M}$
  insert $\overline{J}_i \rightarrow \mathbf{J}$
end for
return J

and the estimate from the entire sample set $\overline{X}_n$ we can estimate the bias and the variance

$$\text{mean} \qquad \mu_J = \frac{1}{M}\sum_i \overline{J}_i$$

$$\text{variance} \qquad \sigma_J^2 = \frac{1}{M-1}\sum_i \left(\overline{J}_i - \overline{X}_n\right)^2$$

$$\text{bias} \qquad \Delta_J = \overline{J} - \overline{X}_n$$

and can calculate confidence intervals as before, where we have split the entire samples set into subsequent sample sets for evaluation. The offsetting of samples considered per jackknife sample $\overline{J}_i$, as performed in the algorithm 1.2, intends to eliminate a possible estimation degradation due to correlation. If correlation among long sample sequences exists, jackknifing yields tighter confidence intervals compared to the method of subsequent sample sets discussed earlier.

Bootstrapping

is a computer based method of statistical inference. It gives direct appreciation of variance, bias, coverage and other probabilistic phenomena related to the evaluation of sample sets [37]. The key principle of the bootstrap is to provide a way to simulate repeated observations from an unknown population using a single sample as basis. Bootstrapping provides a way to account for the distortions caused by the specific sample that may not be a good representative of the population. It also provides some sense on the variability of the estimate that we have computed, and this is precisely what we present using confidence intervals.

The bootstrap provides a conceptually simple computer algorithm to construct confidence intervals from real sample sets. In contrast to the abstract mathematical muddles of direct confidence interval construction using table 1.3, we need nothing more than the sample set and some computation power to get comparably accurate confidence interval bounds. The random draws $\mathbf{S}[x]$ from the real samples in $\mathbf{S}_N$ are performed with replacement, and that the bootstrap sample sets used to calculate the bootstrap estimates $\overline{B}_i$ have the same size N as the real sample set $\mathbf{S}_N$. Because we allow to draw the same sample more than once, the bootstrap sample sets are different form the real and other bootstrap sample sets, even though they all have the same size.

However, bootstrap sample sets cannot provide more information on the process that caused the real sample set than there is information contained in the original sample set. The bootstrap only reveals likely contained information presuming that samples are independent and identically distributed. The achieved set $\mathbf{B}$ of bootstrap estimates $\overline{B}_i$ provides an estimate of the shape of the distribution of the sample mean $\overline{X}_n$, from which we can answer questions about how much the sample

Algorithm 1.3 bootstrap(sample set $\mathbf{S}_N$, confidence level L^C, M)

sorted set $\mathbf{B} = \{\}$
for $i = 1 .. M$ **do**
  $m = 0$
  for $j = 1 .. N$ **do**
    $x = random[1 .. N]$
    $m = m + \mathbf{S}[x]$
  end for
  $\overline{B}_i = m/N$
  insert $\overline{B}_i \rightarrow \mathbf{B}$
end for
with $k^{(-)} = \frac{1}{2}(1 - L^C)M$ and $k^{(+)} = M - k^{(-)}$ we get
  $\overline{X}^{(-)}_{L^C} = \mathbf{B}[k^{(-)}]$ as lower confidence interval bound
  $\overline{X}^{(+)}_{L^C} = \mathbf{B}[k^{(+)}]$ as upper confidence interval bound
return $\left\{\overline{X}^{(-)}_{L^C}, \overline{X}^{(+)}_{L^C}\right\}$

mean varies.

mean	$\mu_B = \frac{1}{M}\sum_i \overline{B}_i$
variance	$\sigma_B^2 = \frac{1}{M-1}\sum_i \left(\overline{B}_i - \overline{X}_n\right)^2$
bias	$\Delta_B = \overline{B} - \overline{X}_n$

However, a sufficiently large number M of bootstrap samples is required to achieve trustworthy confidence interval bounds B_{k^-} and B_{k^+}. $M \geq \frac{100}{(1-L^C)}$ is advised, and commonly $M = 1000, 10000, 100000$ is used for $L^C = 0.9, 0.99, 0.999$, respectively. Effectively, this can be done only using a computer.

Monte Carlo sampling

represents a compromise between approximate randomization and a complete evaluation of all permutations of the available samples within a samples set. The Monte Carlo approach uses a specified number of randomly drawn permutations of the samples comprising a trace or samples set. If the number of drawn permutations is sufficiently large, the Monte Carlo approach tends to outperform approximate randomization based on intuitively selected sub-sets.

Monte Carlo methods follow a pattern:
(a) define the domain of possible scenarios (limits, constraints)
(b) generate random scenarios using a probability distribution over the defined domain
(c) evaluate the generated scenarios (one-by-one, iteratively, recursively)
(d) aggregate the individual results per scenario into general statistical results

In the context of communication system evaluation, the Monte Carlo methods are often used to generate random user populations and user states. The network performance is then evaluated for a given number of randomly selected scenarios. If unsatisfactory results are found the network design is adjusted to better serve the critical scenarios found, and the process is repeated, until a sufficient number of randomly selected scenarios shows no flaws. It is than assumed that for all possible scenarios the performance will be similar to that found for the evaluated scenarios, mean and variance thereof. For more details please refer to the specific literature on the general method.

1.6 Outline and hypothesis

Having sketched the history of communication networks, the basic principle of the exemplarily studied network technology, the problem of performance evaluation, and the basic theory on Markov chains and distribution functions, we now introduce the targets and concepts that cling together the somewhat diverging topics and studies presented in the chapters of this thesis. In the end, we hope that the bigger picture, the hypothesis and the objectives, become apparent.

1.6.1 Traffic and system models

In the second chapter we continue what has been introduced in section 1.5, the modelling of arrival and system behaviour by distribution functions. Chapter 2 comprises a survey on popular distribution function families, listing and discussing their definition and specific properties.

Chapter 3 continues what has been introduced in section 1.4, the modelling and analysis of subsystem performance by means of Markov-chain models and other techniques. We review and explain different analysis options with different queueing model examples, introducing both, the different models and the different options to analyse them.

Arrival and system behaviour models

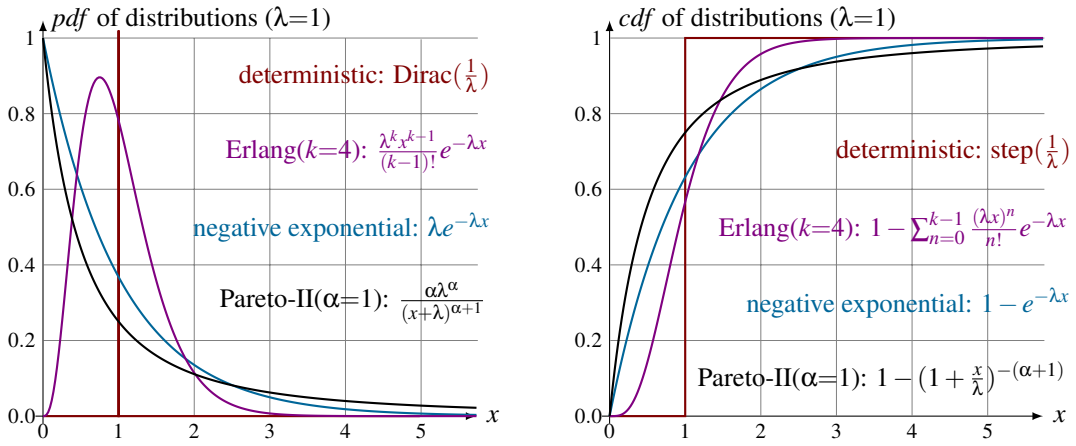


Figure 1.28: Continuous distribution function types: *pdf* and *cdf* examples

Starting with the most popular distributions family based on the negative exponential distribution, we show how the well known and also quite complex distributions can be defined as *phase type distributions*. It is said that the phase type distributions are dense in the field of all positive-valued distributions and thus, it can be used to approximate any positive-valued distribution. However, the number of required phases may approach infinity, which may be infeasible in practice. Completing the family, we present the *Markov arrival process* (MAP) and show how this can be designed according to system characteristics. The simplest non trivial variants thereof are the *interrupted Poisson process* (IPP) and the *Markov modulated Poisson process* (MMPP).

Because there is evidence that in practice some systems are characterised by behaviours that do not well fit into the negative exponential distribution family, we present also the somewhat artificial *deterministic process* on one side and on the other, some *power law* based distributions, in particular the *Pareto family* and the *Beta distribution*. Such distributions are commonly difficult to handle analytically, but for simulation studies they can be implemented with negligible extra effort. Practically, the deterministic process results from any distribution function when we let the variance become zero. For analytic, function based evaluation, we can use the Dirac and the step function to represent the deterministic *pdf* and *cdf*, respectively.

Queueing system models

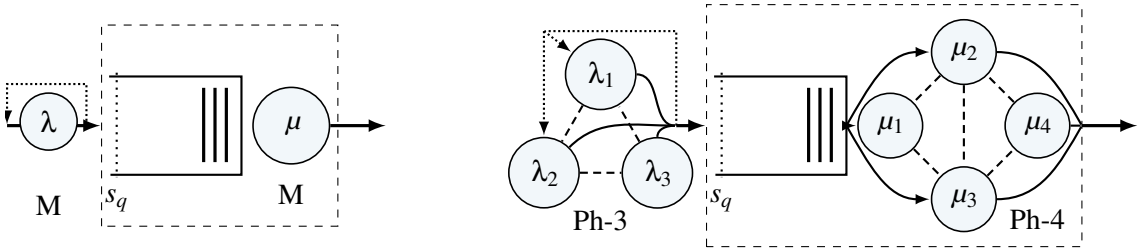


Figure 1.29: M/M/1/s and Ph/Ph/1/s queueing systems

We start with explaining the Kendall notation and how we extend it. Than we present the analytic approach to *infinite, finite, and queue-less systems*. As far as possible also non-Markovian arrival and service processes are exemplary considered, as well as the *Engset model* for finite customer populations. In particular we present the numeric calculation options based on state transition matrices, being the *Matrix Geometric Method* (MGM) and the *Matrix Analytic Method* (MAM), which are particularly handsome to numerically analyse *Ph/Ph/...* queueing systems.

The core intention of chapter 3 is to show and to delve into the plurality of models, their elegance, and the related solving options. This shall prepare the grounds for the multi-queue multi-flow models, in particular the methods to handle these in a similar and efficient way, which are the core topic of chapter 4 exemplifying the usage of these systems to model real network components on their own and meshed into networks.

Chapter 3 concludes with a step-by-step discussion of the simulation routines applied to generate the simulation results used to validate the analytic results. Options and limitations of the used method are in detail presented. The complete simulation core, excluding the random number generation and other not directly relevant parts, is presented in addendum A.I.2.

1.6.2 Traffic handling network elements

In section 4.1 we extend and apply the models presented in chapter 3 to study different scheduling mechanisms, particularly addressing the introduced delay, whereas section 4.2 puts the focus on the loss rates of the mechanisms used to resolve or avoid congestion. With both also fairness issues are addressed and the evaluated methods are compared in that respect.

Network element models for shared resources

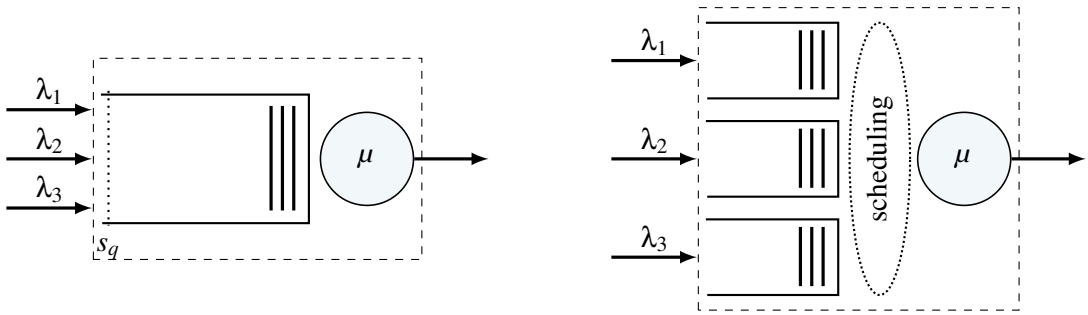


Figure 1.30: Single-queue and multi-queue resource sharing

First we address the prioritisation schemes commonly used to protect some flows from the effects of contention. We show that prioritisation cannot speed up the network resources, and that every advantage we achieve for one flow is a disadvantage for the less prioritised flows. This problem is addressed next, where we evaluate the fair queueing mechanism and its weighted sibling, the *weighted*

fair queueing WFQ mechanism. This mechanism is today most widely used, and therefore we intensely analyse it by approximating it via a random queue selection scheme that shows comparable performance.

Network element models for congestion management

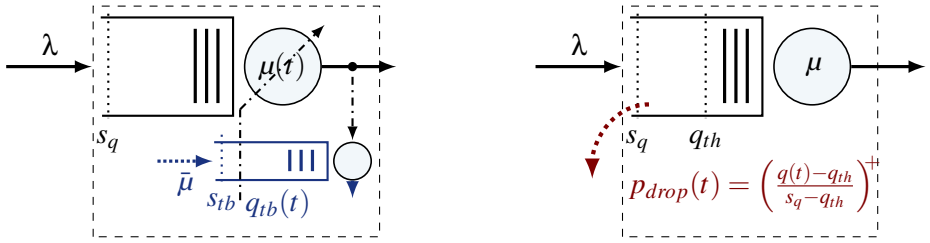


Figure 1.31: Insertion binding (TB) and congestion avoidance (RED)

Hop-by-hop routing allows immediate insertion of the traffic at the source without any delay. This *send-and-forget* paradigm causes that there is no systematic coordination of the load insertion. This implies an inevitable potential for temporary congestion here and then at changing locations throughout the network. The simple method to keep the thereby caused losses low, is over-dimensioning the resources and statically limiting the access, meaning the insertion rate per source. Latter is commonly performed by a so called *token bucket* (TB) mechanism, which allows a certain peak load that may be inserted for a short duration and a much lower average rate that is allowed in the long term.

To better utilize the resources without causing excessive loss rates we need to actively manage the insertion rate at sources. This is performed by a *transport control protocol* (TCP). Note that initially IP has been proposed with this mechanism integrated [38]. Today, we use IP with a variety of transport control schemes. Most respond to lost packets by a quick reduction of the insertion rate, and slowly increase the insertion rate while no losses occur. However, not all flows respond to losses, and those that do respond may be served unfairly poor. In this context we evaluate the *random early detection* (RED) method and its weighted relative (WRED), which are essential for TCP to perform well. We develop Markov chain models, calculate their performance, and highlight their impact on passing traffic flows. Of particular interest to us is WRED, where in theory we could autonomously adjust the weighting factors in a way that provides the features required to realise the network components outlined in chapter 4.4 for a distributed, self managed, QoS provisioning network architecture.

1.6.3 Operating networks of individual elements

A difficult to respond to critic is that we need to simplify many things in order to analyse the network performance. Practically, the plurality of elements causes that the number of possible network architectures, meaning compositions of these elements to build functional networks, is nearly unlimited. Thus, we concentrate on chains of queueing systems because these can be modelled by a joint system state matrix. The other approach is to model an entire path across a network as single server queueing system with a rather guessed service distribution. This simplification allows us to use elementary queueing models to evaluate the mechanisms present at terminals in a stand alone fashion. Studies of entire network topologies have not been performed, primarily because of the lack of relevance if the topology is too heavily simplified. In addition, a transparent simulation environment based on the queueing models covered in this thesis could not be developed in reasonable time.[6]

[6] see addendum A.II for details on the envisaged simulation environment.

Network element chains

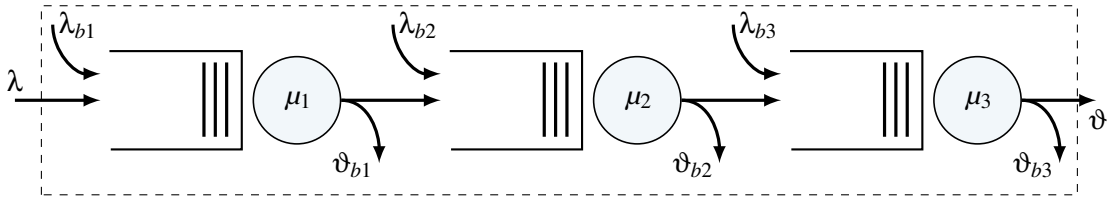


Figure 1.32: Chain of queueing systems with local background traffic

Cascades of different network elements along paths are potentially influenced by all traffic flows present in the network. Studying them individually, assuming independent and identically distributed background traffic is an option, but clearly a simplification. Figure 1.33 shows a short end-to-end connection for three traffic classes passing one core network node only. Still, this connection contains

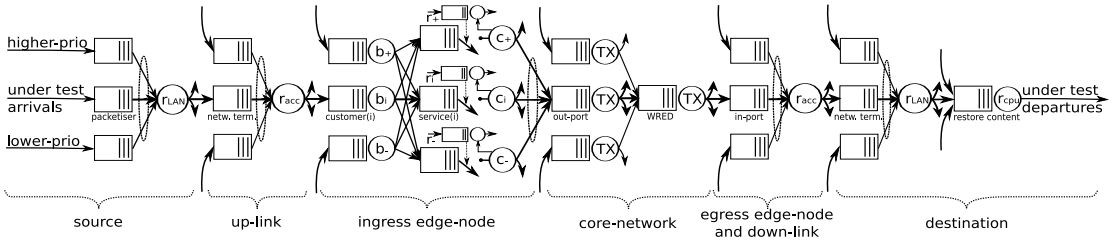


Figure 1.33: Chain of queueing systems contributing to a connection

twenty six queues and thus causes a 26-dimensional queueing model. Every additional core node adds four more queues, and every additional traffic class scales up the dimensionality. Evidently, this hardly can be solved efficiently by Markov chains: for queues of size ten we would get more than 11^{26} states and an accordingly huge equation system to solve. Most software tools are for today not capable thereof or it would last ages to get results.

End-to-end traffic management

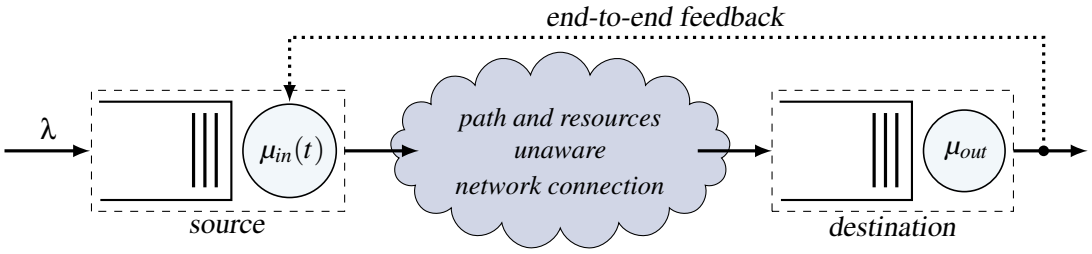


Figure 1.34: End-to-end transport control based on destination's feedback

In section 4.2 we introduce the *random early detection* (RED) queue control mechanism and mentioned that this achieves congestion avoidance when combined with a loss responsive *transport control protocol* (TCP). The feedback to sources is delayed by one *round trip time* (RTT) and therefore the response needs to be pessimistic. The few but randomly distributed early drops caused by RED are essential to avoid augmented reaction of many sources at the same time. First we evaluate the effect of RED and weighted RED on the performance per node, and later discuss how the known *pumping* issue is mitigated by RED. Finally, we evaluate ingress binding and flow shaping options, to see exemplarily to which extent these can further improve the network throughput.

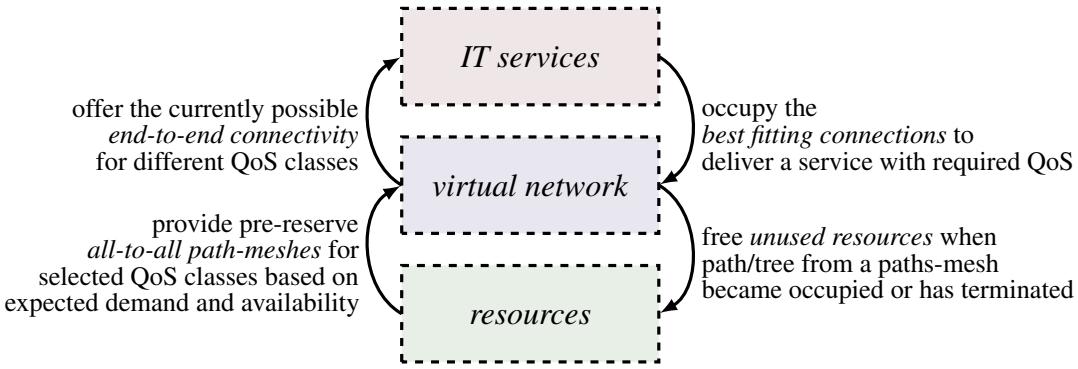


Figure 1.35: QoS sensible support of dynamic connectivity based on technology independently provided pre-reserved capacity across autonomous network sections

Network operation hierarchy

Finally, concluding the thesis, we address network management and propose the operation approach sketch in figure 1.35, which combines resource provisioning and QoS delivery in a technology agnostic fashion. Resources commonly provide static bit-rates, which are either used or not. More flexible approaches have been proposed recently, for example *flexible grid WDM* [39] and flexibly utilised GMPLS in the context of *polymorphous agile transparent optical networks* (PATON) [40]. However, to be scalable, we need an intermediate system that on demand combines the provided capacities between nodes to end-to-end connections. Today, this is performed perfectly neutral by IP, and across IP-domains by the *border gateway protocol* (BGP) [RFC4271]. These work solely on the packet layer. If the qualities of other switching technologies shall be integrated, a less technology centric resource broker is required in order to trade the provided QoS fittingly to all the dynamically risen transmission demands of customer driven IT-services.

We therefore introduce in section 4.4.4 a *virtual network management* that technology independently selects from the connectivity advertised by autonomous network sections, being domains and technologies, a route that fits the requesting service. The autonomous network sections are assumed to heavily overlap and to always advertise to the virtual network management instantly accessible connectivity at different QoS levels. Thereby, a lean and solely demand driven management is envisaged.

1.6.4 An outlook beyond the covered issues

Once we have models for all common network elements it is possible to create a hybrid simulation environment where the load is simulated but the network elements' performance is calculated analytically using queueing system models. The instances of such a simulation environment as well as the communication among these is sketched in figure 1.36. Basically, the simulation replaces the

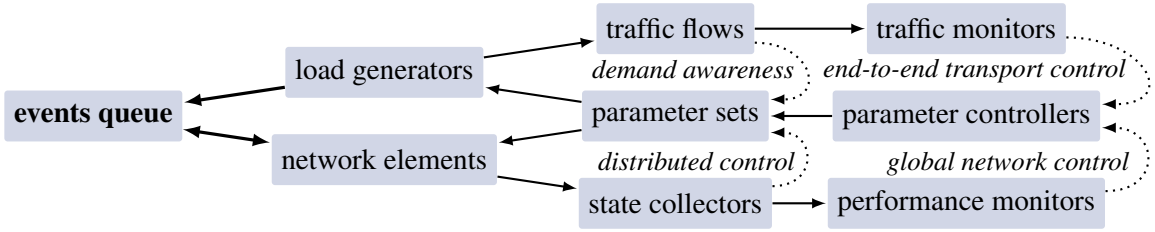


Figure 1.36: Simulation architecture for networks of queueing systems

cloud in figure 1.34 by meshed network element models. A more detailed sketch of the different simulation objects can be found in the addenda, A.II.

Meshed control strategy

Based on the studies presented in this thesis the long term target is to enable novel network operation schemes alike what had been intended for the *next generation network* (NGN) [2]. First and essential to realise an NGN compliant network architecture is the ability to deliver guaranteed *quality of service* (QoS) to the demanding services.[7]

In figure 1.36 we sketched the common control loops that may be utilised jointly to deliver a requested QoS. First there is the *demand awareness* loop that allows to set-up resources for a specific flow. This is for example realised by setting-up or re-assuring the current per hop behaviours for certain traffic classes along label switched paths (on demand signalling). To its right we find the end-to-end control loop that for example TCP implements. Based on the destination's feedback the source's parameters can be changed. This may include a change of the insertion rate, the used path, the traffic class, the priority level, and so on. Below that we have the global network control loop commonly implemented by some network management system. It commonly relies on an assumed complete knowledge about all resources it controls, and uses off-line optimisation tools to calculate better resource parametrisation (long term optimisation). Finally, we find on its left the *distributed control* loop. This is for example realised by RED, though not adaptive to per flow quality demands.

Localised flow management

To our understanding a decentralised scheme is more scalable and sustainable than any centralised approach. However, the decentralisation needs not be put to the extreme, network processors may be used in a decentralised fashion to quickly estimate the locally required adjustments for entire groups of network elements. The target remains to granted end-to-end QoS per flow, independent of the network elements passed and the competition for resources locally present.

The in our view natural approach to local management is shown in figure 1.37. First the flows

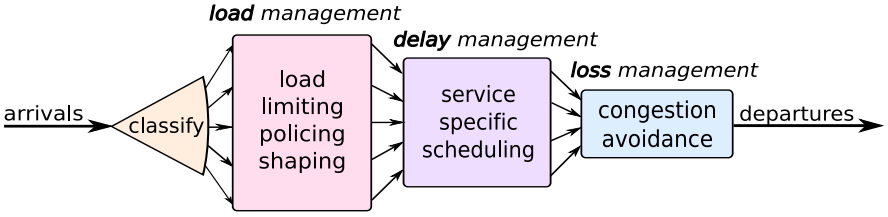


Figure 1.37: Locally available control options include load-, delay-, and loss-control

need to be identified and their demands extracted. Than the load they cause should be monitored and in case necessary adjusted. Only after the individual loads have been determined the scheduling can be decided. Finally, if all the previous steps did not prevent congestion right away, some means to discard excessive load need to be provided for stability reasons.

However, to perform the above in an end-to-end perspective, the nodes need to communicate with their peers, at least the neighbouring peers. The therefore required distributed local control loops are to our knowledge least exploited in practice.

The interactions are critical and therefore is controlling a network with hundreds of interdependent local control loops a seemingly far too complex problem. We fully agree that a deterministic approach leading to a precise and perfectly predictable control scheme is not an option. However, novel rather fuzzy and strictly conservative although imprecise schemes promise robust and at the same time sufficiently simple control strategies.

[7] Anything not yet realised is of interest for science. Thus, we look on NGN rather open minded. Things change and what today may seem unrealistic can become state-of-the-art some time if it does not vanishes into thin air.

Distributed routing control

Actually, resource competition is the reason for most QoS degradation. Therefore, *smart load balancing* is the most effective approach to QoS delivery. We envisage a distributed routing control scheme, which heavily utilizes inter-node signalling. For example, the ants principle could be used to introducing dynamic adjusting of the routing tables based on locally accumulated information from passing ants that collected the relevant information along their individual journeys.

Ant types and their rules

three types

- **worker:** follow the path used by the most –> follow the masses
- **explorer:** choose the least popular routing –> find new paths
- **individualist:** randomly choose from known paths –> preserve alternatives

three rules

- **searching:** choose path according to the ant's type –> operate the system
- **returning:** add pheromones if successful –> update path information
- **blocked:** convert into *explorer* to circumvent blockage –> become creative

Note, ants never report about a blockage because that does not solve the problem and thus moaning would be counterproductive. If they return, they have survived and thus must have found a path around the problem, which the other ants now follow.

Future-proof network paradigms

Because electronic packet switching is not energy efficient if the packet size becomes very small compared to the transported loads, we need to consider alternative network technologies to be sustainable. Potential candidates are for example *optical burst switching* (OBS), if we rigorously ignore what has been proposed thereon since the exaggerated popularity of [41] has torpedoed the creative process, or more generally the *flow transfer mode* (FTM), introduced in [11] and roughly sketched in figure 1.38. In order to correctly position these, we should compare them with both,

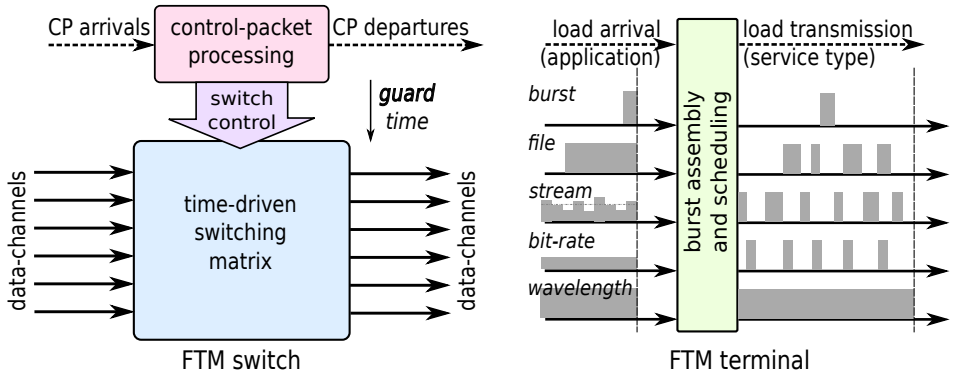


Figure 1.38: Alternative all-optical realisable network technology – FTM

optical time division multiplexed (OTDM) circuit switching and *optical packet switching* (OPS). OBS based on [41] fails the effectiveness check [42].

In figure 1.39 an OBS connection from an ingress edge node across two core nodes to an egress edge node is depicted as chain of queueing and loss systems. The node at the end with the service rate μ_d is the burst de-assembly processor, not a burst handling switch or alike. The egress node

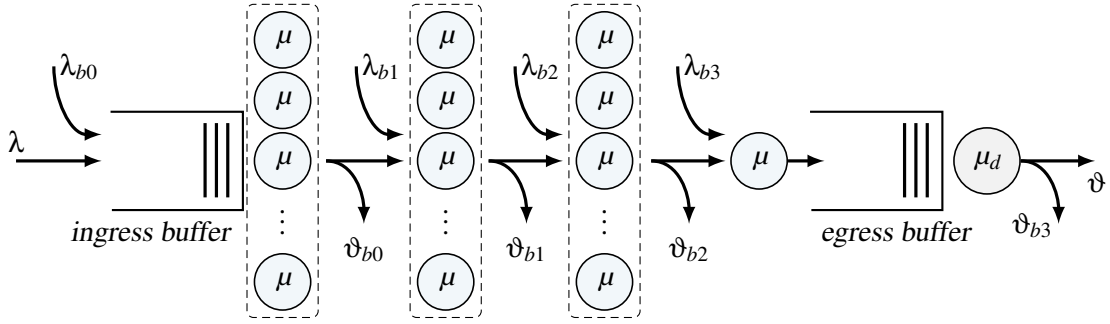


Figure 1.39: Burst switched connection across two core nodes

reduces in the model to a single server, being the switching matrix port connected to the optics-to-electronics (OE) converter available for bursts destined to this node. This port needs to be modelled, because two concurrently arriving burst cannot be received if only one OE converter is foreseen. For more OE converters the number of egress servers increases accordingly. The mixture of queueing and loss systems results from the in principle buffer less operation, and the fact that at ingress and egress nodes we evidently need to handle bursts in electronics and thus can store bursts as long as required.

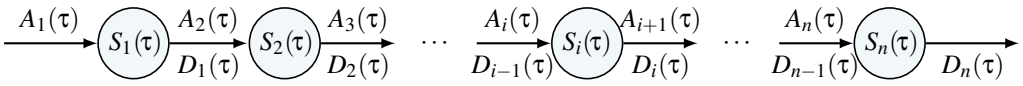
Smart channels – the long term vision

The core feature to end-to-end QoS is in our view decentralised control and quality driven hop-by-hop routing. Local adjustments provide the option to responsively change the per hop behaviour when necessary, but demands the option to influence the traffic treatment along the remaining downstream path in order to compensate the local change. In the end, this reveals an alternative development path toward autonomous, quality controlled packet switched networks: *smart communication channels* based on dynamically adjusted, autonomously operated, flow aware, inter-communicating network elements that establish swarm behaviour in order to jointly provision all the dynamically required channels in parallel.

However, this is a long path to go. A single thesis can hardly cover all issues to be solved along this path. The provided survey and models of key elements represents a first step only. Recently, the development seems to focus on rather static options to reach end-to-end QoS. However, the author of this thesis is convinced that in the long term these recent approaches will dissolve beneficially into fully decentralised schemes.

2 Traffic and system models

In this chapter we introduce stochastic processes and concepts to join these in order to get mathematical models with characteristics that conform to realistic assumptions on data traffic arrival and propagation hop-by-hop as well as end-to-end. The traffic arrival and propagation end-to-end refers to the load arriving to the resources that establish a communication connection. We refer to it as the *arrival process* $A_i(\tau)$, because it defines the incoming load that the systems that control the local resources need to handle hop-by-hop. The hop-by-hop serving we refer to as the *service process* $S_i(\tau)$, because it is determined by the protocols locally applied to handle the passing traffic load. The output of each such system we refer to as *departure process* $D_i(\tau)$. For chained systems the input of



the following system equals the output of the previous system. For meshed systems the input to a system is the aggregation of the outputs of all connected systems that can forward traffic load to it.

Technically controlled systems are in practice rather not stochastic. However, the variability of the possible input parameters and the cascaded control mechanism typically involved in modern processor controlled mechanisms, cause a degree of freedom that hardly can be considered in its entirety. Only for specific situations we can state exemplary responses precisely. Therefore, if we can define a stochastic process that yields approximately the same response for any specific situation, it can be used to model the technical system without loss.

To be mathematically traceable we intend to model the characteristic with a maximally simple distribution function. We will see in the subsequent chapters that even for negative exponentially distributed traffic end-to-end connection models cannot be solved analytically, without additional simplifications. Therefore, the more precise we model the traffic the more we will need to simplify the model, which deteriorates the precision of the model. A good balance between the accuracy of the traffic model and the system simplification needs to be achieved.

Process models are based on the combination of well known random processes. Most famouse and commonly used with communications are the *Poisson*, the *Pareto*, and the *Markov arrival process*. These show specific properties and may be combined to model more complex behaviours. In the following we first introduce these and some more well know processes and their specific properties, before we proceed to the adjusting of these processes to match design specific system behaviours or recorded traces thereof. Later on we split the local serving into a *waiting phase* and a decoupled service process. This adds the option to explicitly analyse the queueing of incoming load and reduces the complexity of the service process to the serving of individual load chunks. It leads us directly to the well known *queueing systems* and their analysis. With queueing systems we strictly use λ for arrival rates and μ for service rates. Until then, we look at singular processes only, and defining special meanings for λ and μ is not possible, because the presented process may be used to model an arrival process or a service process likewise. Thus, in the following we may use λ and μ interchangeably.

2.1 Markovian processes

The term *Markov process* stands for all processes that fulfil the Markov property of being *memoryless*: for a defined current state the future outcome is independent from whatever happened in the past. More particular, the future evolution of the process is independent of how that current state has been reached. Only the current state may, and commonly does, influence the near future. This property is fulfilled by the negative exponential distribution, and therefore any uncorrelated combination of negative exponential distributions fulfils the Markov property as well.

With *Markov chains* such *renewal processes* can be modelled. Each *phase* in such a chain adds a single negative exponential component that itself can be modelled by a Poisson process. The *linkage* of the intermediate phases defines the resultant properties, and for many regular combinations known random processes result.

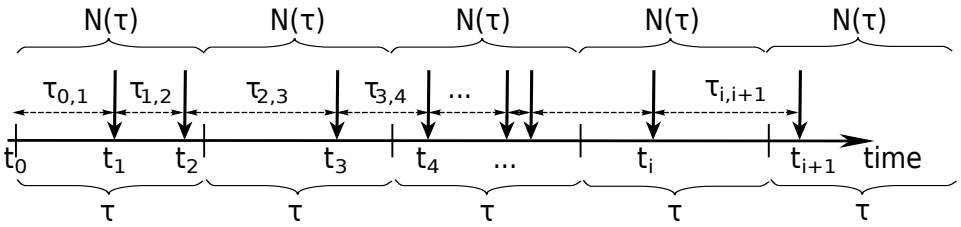
Markovian distributions and their abbreviations

- M *Markovian*: results from a single Poisson process – memoryless
- Geo *Geometrical*: discrete pendant to M – memoryless
- E_k *Erlangian*: series of k equal M's – smoother than M
- H_k^- *Hypo-exponential*: series of k M's – generalised Erlang $E_k(\vec{\lambda})$
- H_k^+ *Hyper-exponential*: k M's in parallel – more bursty than M
- Cox_k *Coxian*: tapped series of k M's – mixed generalised Erlang $E_k(\vec{p},\vec{\lambda})$
- Ph_k *Phase-type*: k M's in a mesh – most general renewal process
- MAP *Markov arrival process* – generalised Poisson process

Note that renewal processes are memoryless only in respect to subsequent departure events. The internal transitions in between departure events commonly depends on the recent past, meaning the *random walk* that determines the chain of intermediate states passed to reach the current state since the last event occurred. The MAP is memoryless only in respect to subsequent departures related to the same transition among the phases, *the Markov chain*, that defines the process. The departure events within a chain of events that occur in between two subsequent departures related to the same transition are not independent.

2.1.1 The Poisson process

The *Poisson process* is one of the most common, at the same time most special random processes. We briefly introduced it in section 1.5.1 for its importance, and repeat it here for completeness of chapters. The Poisson process generates negative exponentially distributed inter-event times $\tau_{i,i+1} = t_{i+1} - t_i$,



with uniformly distributed event occurrence times t_i, and Poisson distributed number of events $N(\tau)$ per time interval τ.

$$\mathrm{P}[N(\tau)=n] = \frac{(\lambda\tau)^n}{n!}\, e^{-\lambda\tau} \tag{2.1}$$

All events are independent and identically distributed (i.i.d.), the inter arrival time $\tau_{i,i+1}$ neither depends on the event's-index i nor the time since the last arrival $\tau_{i,now}$.

$$P[\tau_{i,i+1} > \tau_{i,now} + \tau | \tau_{i,i+1} > \tau_{i,now}] = P[\tau_{i,i+1} > \tau] = P[N(\tau) = 0] = e^{-\lambda\tau} \tag{2.2}$$

Obviously, the probability that the inter-event-time $\tau_{i,i+1}$ exceeds a certain duration τ must equal the probability that in an interval with length τ no events occur.

$$P[\tau_{i,i+1} > \tau] = P[N(\tau) = 0] \tag{2.3}$$

This property provides an important relation, which is frequently used to swap between probabilities of event numbers and event times. It also relates to the important property that, due to the uniform distribution of occurrences within any time interval, the proportion of occurrences at different system states equals the probabilities of these states. This property is known as the **PASTA** law (**P**oisson **a**rrivals **s**ee **t**ime **a**verages).

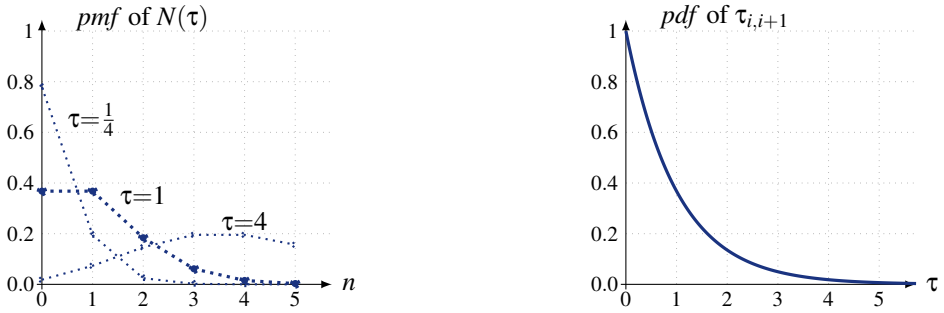


Figure 2.1: Event-count and inter-event-time distribution for $\lambda = 1$

Note that the *Poisson distribution* is a discrete distribution that depends on both, λ and τ. The dotted lines in figure 2.1 are shown for better visualisation only; $N(\tau)$ is not existent for $n \notin \mathbb{N}^0$. The negative exponentially distributed inter-event-time distribution shown on the right is continuous and depends on λ only. The negative exponentially distributed inter-event times alone are not sufficient to constitute that a Poisson process generated them. However, we may use the Poisson process whenever we want to generate negative exponentially distributed events.

2.1.2 Phase-type distributions

Phase-type distributions result if we combine individual phases, where each intermediate phase is described by a negative exponential holding time distribution. Any number of intermediate phases may be combined in any structure. Only the holding time distributions of phases have to be independent.

Phase-type distributions

- result from k interrelated Poisson processes, $k \geq 1$ represented by
  - a *subgenerator matrix* $T_{[k\times k]}$,
  - an *entry-vector* $\alpha_{[k]}$, and
  - a resultant *exit-vector* $t_0 = -T1$,
- which together specify the
  - *pdf*: $$f_{Ph}(x) = \alpha e^{Tx} t_0, \tag{2.4}$$
  - *cdf*: $$F_{Ph}(x) = 1 - \alpha e^{Tx} 1, \tag{2.5}$$
  - *moments*: $$E[{X_{Ph}}^n] = (-1)^n n! \alpha T^{-n} 1. \tag{2.6}$$

- The generator matrix is defined by

$$Q = \begin{bmatrix} T & t_0 \\ 0^T & 0 \end{bmatrix} = \left[\begin{array}{cccc|c} t_{11} & t_{12} & \cdots & t_{1k} & t_{0,1} = -\sum t_{1j} \\ t_{21} & t_{22} & \cdots & t_{2k} & t_{0,2} = -\sum t_{2j} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_{k1} & t_{k2} & \cdots & t_{kk} & t_{0,k} = -\sum t_{kj} \\ \hline 0 & 0 & \cdots & 0 & 0 \end{array}\right]_{[k+1\times k+1]} \tag{2.7}$$

  where $\sum_{j|j\neq i} t_{ij} \leq 1 \ \forall_i$ is required to assure valid exit rates $t_{0,i} = p_{0,i}\lambda_i$.

The phase-type distributions are in theory dense in the field of all positive-valued distributions. Thus, they can represent any such distribution. However, they are light-tailed and although heavy-tailed can be approximated with increasing number of phases, it remains in practice always an approximation, because for perfect heavy-tail fitting an infinite number of phases would be required.

Special phase-type examples

Being defined as combination of several phases it is evident that if we only specify one phase we get the negative exponential distribution. Similarly, for special combinations we get the well known distributions listed here.

- M (Markovian)

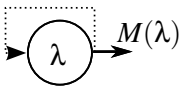


Figure 2.2: Poisson generator

$$T = \begin{bmatrix} -\lambda \end{bmatrix}_{[1\times 1]}, \ \alpha = \begin{pmatrix} 1 \end{pmatrix}_{[1]} \Rightarrow Q = \left[\begin{array}{c|c} -\lambda & \lambda \\ \hline 0 & 0 \end{array}\right] \tag{2.8}$$

Negative exponential distribution

- *pdf*:

$$f_M(x) = \lambda e^{-\lambda x}, \quad x \geq 0$$

- *cdf*:

$$F_M(x) = 1 - e^{-\lambda x}, \quad x \geq 0$$

- *moments*: $\mathrm{E}[X_M^n] = n!\lambda^{-n}$

$$\mathrm{E}[X_M] = \frac{1}{\lambda} \qquad Var(X_M) = \frac{1}{\lambda^2}$$

- *coefficient of variation*:

$$c_{X_M} = \frac{\sqrt{Var(X_M)}}{\mathrm{E}[X_M]} = 1$$

- E_k (Erlangian)

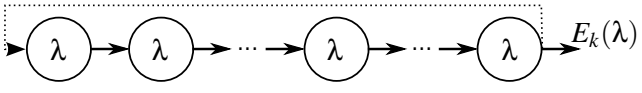


Figure 2.3: Erlang distribution generator

$$T = \begin{bmatrix} -\lambda & \lambda & 0 & \cdots & 0 \\ 0 & -\lambda & \lambda & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\lambda & \lambda \\ 0 & \cdots & 0 & 0 & -\lambda \end{bmatrix}_{[k\times k]}, \alpha = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}_{[k]} \tag{2.9}$$

Erlang-k distribution

- *pdf*:

$$f_{E_k}(x) = \lambda \frac{(\lambda x)^{k-1}}{(k-1)!} e^{-\lambda x}, \quad x \geq 0$$

- *cdf*:

$$F_{E_k}(x) = 1 - \sum_{i=1}^{k} \frac{(\lambda x)^{i-1}}{(i-1)!} e^{-\lambda x}, \quad x \geq 0$$

- *moments*:

$$\mathrm{E}[X_{E_k}] = \frac{k}{\lambda} \qquad Var(X_{E_k}) = \frac{k}{\lambda^2}$$

- *coefficient of variation*:

$$c_{X_E} = \frac{1}{\sqrt{k}} \leq 1$$

- H_k (Hyperexponential)

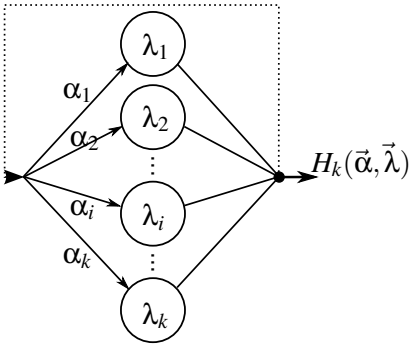


Figure 2.4: Hyperexponential distribution generator

$$T = \begin{bmatrix} -\lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & -\lambda_2 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\lambda_{k-1} & 0 \\ 0 & \cdots & 0 & 0 & -\lambda_k \end{bmatrix}_{[k\times k]}, \alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{k-1} \\ \alpha_k \end{pmatrix}_{\sum \alpha_i = 1} \tag{2.10}$$

Hyper-exponential distribution

- *pdf*:

$$f_{H_k}(x) = \sum_{j=1}^{k} \alpha_j \lambda_j e^{-\lambda_j x}, \quad x \geq 0$$

- *cdf*:

$$F_{H_k}(x) = \sum_{j=1}^{k} \alpha_j \left(1 - e^{-\lambda_j x}\right), \quad x \geq 0$$

– *moments*:

$$\mathrm{E}[X_{H_k}] = \sum_{j=1}^{k} \frac{\alpha_j}{\lambda_j} \qquad Var(X_{H_k}) = 2\sum_{j=1}^{k} \frac{\alpha_j}{\lambda_j^2} - \left(\sum_{j=1}^{k} \frac{\alpha_j}{\lambda_j}\right)^2$$

– *coefficient of variation*:

$$c_{X_H} = \sqrt{\frac{2\sum \frac{\alpha_j}{\lambda_j^2}}{\left(\sum \frac{\alpha_j}{\lambda_j}\right)^2} - 1} \geq 1$$

For $\lambda_i = \lambda \; \forall_i \; \Rightarrow$ *degenerated case* of a simple Markov distribution $M(\lambda)$.

- Cox_k (Coxian or mixed generalised Erlang)

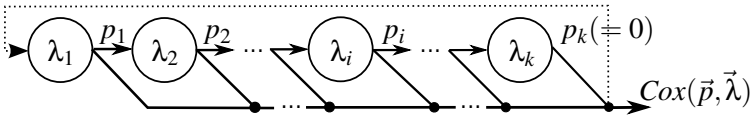


Figure 2.5: Cox distribution generator

$$T = \begin{bmatrix} -\lambda_1 & p_1\lambda_1 & 0 & \cdots & 0 \\ 0 & -\lambda_2 & p_2\lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\lambda_{k-1} & p_{k-1}\lambda_{k-1} \\ 0 & \cdots & 0 & 0 & -\lambda_k \end{bmatrix}_{[k\times k]}, \alpha = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}_{[k]} \tag{2.11}$$

$$\mathrm{E}\left[X_{Cox(\vec{p},\vec{\lambda})}\right] = \frac{1}{\lambda_1} + p_1\left(\frac{1}{\lambda_2} + p_2\left(\frac{1}{\lambda_3} + p_3(\ldots)\right)\right)$$

For $p_i = 1 \; \forall_{i<k} \; \Rightarrow$ *hypoexponential* distribution $H_k^-(\vec{\lambda})$,
also known as *generalised Erlang* distribution $E_k(\vec{\lambda})$.

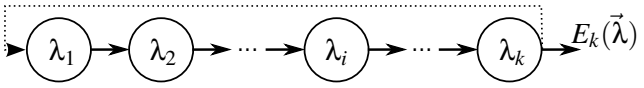


Figure 2.6: Generalised Erlang distribution generator

$$\mathrm{E}\left[X_{E_k(\vec{\lambda})}\right] = \frac{1}{\lambda_1} + \frac{1}{\lambda_2} + \cdots = \sum_{j=1}^{k} \frac{1}{\lambda_j}$$

For $\lambda_i = \lambda \; \forall_i \quad \Rightarrow$ *mixed Erlang* distribution $E_k(\vec{p^*})$ results,
where $p_j^* = (1-p_j)\prod_{i=1}^{j-1} p_i \quad$ for $j = 1 \,..\, k$.

The definition of mixed Erlang and mixed generalised Erlang as parallel-serial combination of phases shown in figure 2.7 is self explanatory and useful for Laplace transformation and calculations in general. Due to the many more states required, it is not efficient to use this to define the sub-generator matrix T. The Cox form uses the least number of phases required to define the same distribution and yields the non-reducible sub-generator matrix T directly.

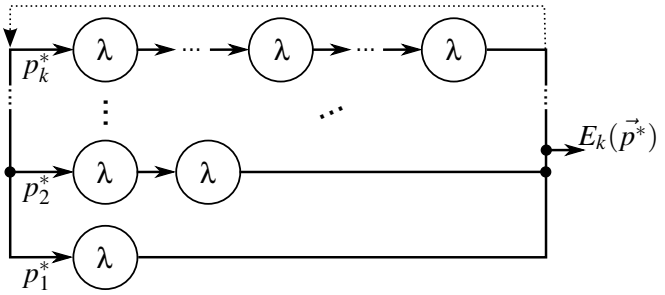


Figure 2.7: Mixed Erlang distribution generator

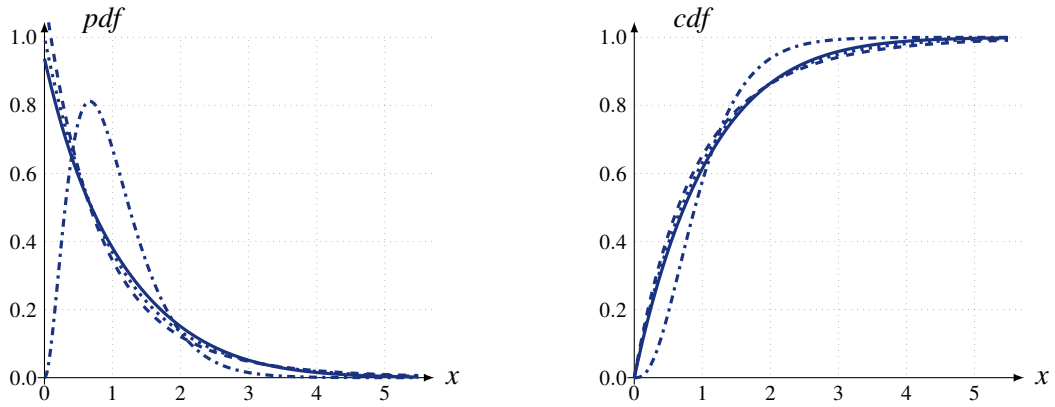


Figure 2.8: *pdf* and *cdf* of $M[\lambda{=}1]$ (dotted), $E_3[\lambda{=}3]$ (dash-dotted), $H_2[\lambda{=}\binom{0.75}{1.5}, \alpha{=}\binom{0.5}{0.5}]$ (dashed), and *homogeneous* $Cox_4[\lambda_i{=}1.875, p_i{=}0.5]$ (solid)

Examples for the above sketched special cases of phase-type distributions are shown in figure 2.8. The distributions parametrisation has been chosen to yield unit mean. For the Cox_4 this has been iteratively achieved up to 4 digits by adjusting its λ_i.

The first five raw moments of the special phase-type examples shown in figure 2.8 are

$$\begin{aligned}
&\mathrm{E}[X_{E_3}] = 1.00, \quad \mathrm{E}\left[{X_{E_3}}^2\right] = 1.33, \quad \mathrm{E}\left[{X_{E_3}}^3\right] = 2.22, \quad \mathrm{E}\left[{X_{E_3}}^4\right] = 4.44, \quad \mathrm{E}\left[{X_{E_3}}^5\right] = 10.37 \\
&\mathrm{E}[X_{Cox_4}] = 1.00, \quad \mathrm{E}\left[{X_{Cox_4}}^2\right] = 1.85, \quad \mathrm{E}\left[{X_{Cox_4}}^3\right] = 4.78, \quad \mathrm{E}\left[{X_{Cox_4}}^4\right] = 15.53, \quad \mathrm{E}\left[{X_{Cox_4}}^5\right] = 60.20 \\
&\mathrm{E}[X_M] = 1.00, \quad \mathrm{E}\left[{X_M}^2\right] = 2.00, \quad \mathrm{E}\left[{X_M}^3\right] = 6.00, \quad \mathrm{E}\left[{X_M}^4\right] = 24.00, \quad \mathrm{E}\left[{X_M}^5\right] = 120.00 \\
&\mathrm{E}[X_{H_2}] = 1.00, \quad \mathrm{E}\left[{X_{H_2}}^2\right] = 2.22, \quad \mathrm{E}\left[{X_{H_2}}^3\right] = 8.00, \quad \mathrm{E}\left[{X_{H_2}}^4\right] = 40.30, \quad \mathrm{E}\left[{X_{H_2}}^5\right] = 260.74
\end{aligned}$$

from which we can directly calculate their coefficient of variation c_X.

$$c_X = \frac{\sqrt{Var(X)}}{\mathrm{E}[X]} = \frac{\sqrt{\mathrm{E}[X^2] - \mathrm{E}[X]^2}}{\mathrm{E}[X]} \quad \Rightarrow \quad c_{X_{E_3}} = 0.57, \; c_{X_{Cox_4}} = 0.92, \; c_{X_M} = 1.00, \; c_{X_{H_2}} = 1.10$$

The *homogeneous Cox* distribution introduced in figure 2.8 is commonly used to model hypo-exponential distributions, because it allows to generate distributions with $c_X \leq 1$. It belongs to the mixed Erlang family (same $\lambda \forall_i$), and is further constraint to geometrically decreasing $p_i^* = p^i$. These restrictions effectively reduce the degrees of freedom to three, being the number of phases k, the rate of these phases λ, and the branching probability p. To match a certain mean value we can choose two, to match mean and variation we may still choose one freely. How λ and p are adjusted to match the intended distribution is presented in section 2.3.

General phase-type definition

The above presented special cases show very sparsely populated generator matrices. In general all transitions between phases may be defined. A possibly completely filled transition matrix defines most generally any phase-type distribution.

- Ph_k (Phase-type)

$$T = \begin{bmatrix} -\lambda_1 & p_{12}\lambda_1 & \cdots & p_{1k}\lambda_1 \\ p_{21}\lambda_2 & -\lambda_2 & \ddots & p_{2k}\lambda_2 \\ \vdots & \ddots & \ddots & \vdots \\ p_{k1}\lambda_k & p_{k2}\lambda_k & \cdots & -\lambda_k \end{bmatrix}_{\sum_j p_{ij} \leq 1}, \quad \alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_k \end{pmatrix}_{\sum_i \alpha_i = 1} \tag{2.12}$$

$\sum_{j|j\neq i} p_{ij} \leq 1$ assures *exit probabilities* $p_{0,i} = 1 - \sum_{j|j\neq i} p_{ij} \geq 0$ and thus *exit rates* $t_{0,i} = p_{0,i}\lambda_i \geq 0$
generator matrix

$$Q = \begin{bmatrix} T & t_0 \\ 0^T & 0 \end{bmatrix} = \left[\begin{array}{cccc|c} -\lambda_1 & p_{12}\lambda_1 & \cdots & p_{1k}\lambda_1 & p_{0,1}\lambda_1 \\ p_{21}\lambda_2 & -\lambda_2 & \cdots & p_{2k}\lambda_2 & p_{0,2}\lambda_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{k1}\lambda_k & p_{k2}\lambda_k & \cdots & -\lambda_k & p_{0,k}\lambda_k \\ \hline 0 & 0 & \cdots & 0 & 0 \end{array}\right]_{[k+1\times k+1]} \tag{2.13}$$

- Ph_5 (Meshed phase-type example)

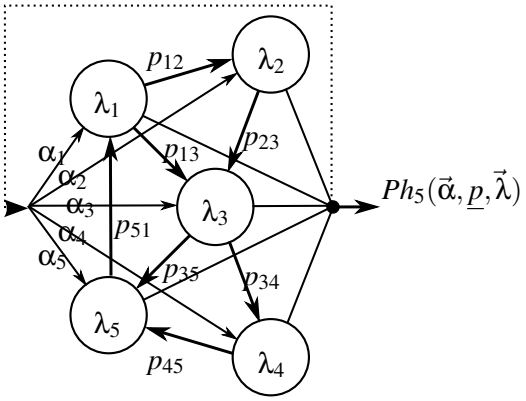


Figure 2.9: Phase type generator example

$$T = \begin{bmatrix} -\lambda_1 & p_{12}\lambda_1 & p_{13}\lambda_1 & 0 & 0 \\ 0 & -\lambda_2 & p_{23}\lambda_2 & 0 & 0 \\ 0 & 0 & -\lambda_3 & p_{34}\lambda_3 & p_{35}\lambda_3 \\ 0 & 0 & 0 & -\lambda_4 & p_{45}\lambda_4 \\ p_{51}\lambda_5 & 0 & 0 & 0 & -\lambda_5 \end{bmatrix}, \quad \alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{pmatrix}_{\sum \alpha_i = 1} \tag{2.14}$$

We note that there are quite many zeros in the T matrix, however, this is only for presentation simplicity. The matrix of a fully meshed generator would be completely filled. The diagonal elements show the inverse of the mean holding time ($\tau_h(i) = \frac{1}{\lambda_i}$). A zero holding time would indicate transient states, and an infinite holding time can result for absorption states only. For steady-state state probability analysis the former can be neglected and the latter may not exist.

To exemplify how the equations 2.4, 2.5, 2.6 for phase type processes can be applied, we need concrete figures. We set

$$\alpha = \begin{pmatrix} 0.10 \\ 0.15 \\ 0.20 \\ 0.25 \\ 0.30 \end{pmatrix} \qquad p = \begin{bmatrix} - & 0.4 & 0.4 & 0.0 & 0.0 \\ 0.0 & - & 0.8 & 0.0 & 0.0 \\ 0.0 & 0.0 & - & 0.4 & 0.4 \\ 0.0 & 0.0 & 0.0 & - & 0.8 \\ 0.8 & 0.0 & 0.0 & 0.0 & - \end{bmatrix} \qquad \lambda = \begin{pmatrix} 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{pmatrix}$$

and get

$$T = \begin{bmatrix} -3 & 1.2 & 1.2 & 0 & 0 \\ 0 & -4 & 3.2 & 0 & 0 \\ 0 & 0 & -5 & 2.0 & 2.0 \\ 0 & 0 & 0 & -6 & 4.8 \\ 5.6 & 0 & 0 & 0 & -7 \end{bmatrix} \quad \text{and} \quad t_0 = \begin{pmatrix} 0.6 \\ 0.8 \\ 1.0 \\ 1.2 \\ 1.4 \end{pmatrix}.$$

Now we can numerically calculate the *pdf* and *cdf* depicted in figure 2.10, using the `expm(T)` function provided by *Octave* to calculate the matrix exponential e^{Tx} in equation 2.4 and 2.5.

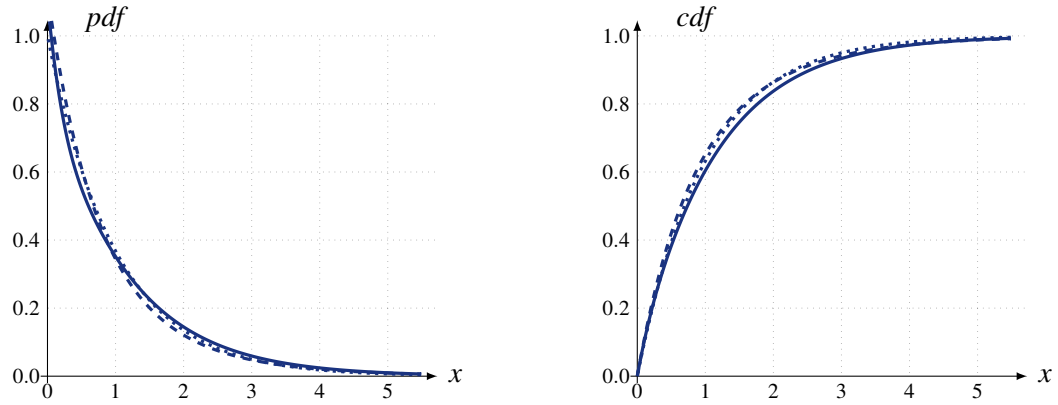


Figure 2.10: *pdf* and *cdf* of Ph_5 example (solid), $H_2[\lambda=\binom{0.75}{1.5}, \alpha=\binom{0.5}{0.5}]$ (dashed) and $M[\lambda=1]$ (dotted)

How to interpret the very similar curves shown in figure 2.10, methods to present them more separated, and conclusions that may be derived from curve shapes, are discussed later on in section 2.3.1. The moments of the example phase-type distribution, calculated using equation 2.6, compared to the moments of the hyperexponential H_2 and the negative exponential distribution, are:

$$\begin{array}{lllll} \mathrm{E}[X_{Ph_5}] = 1.09, & \mathrm{E}\left[{X_{Ph_5}}^2\right] = 2.43, & \mathrm{E}\left[{X_{Ph_5}}^3\right] = 8.18, & \mathrm{E}\left[{X_{Ph_5}}^4\right] = 36.68, & \mathrm{E}\left[{X_{Ph_5}}^5\right] = 205.69 \\ \mathrm{E}[X_{H_2}] = 1.00, & \mathrm{E}\left[{X_{H_2}}^2\right] = 2.22, & \mathrm{E}\left[{X_{H_2}}^3\right] = 8.00, & \mathrm{E}\left[{X_{H_2}}^4\right] = 40.30, & \mathrm{E}\left[{X_{H_2}}^5\right] = 260.74 \\ \mathrm{E}[X_M] = 1.00, & \mathrm{E}\left[{X_M}^2\right] = 2.00, & \mathrm{E}\left[{X_M}^3\right] = 6.00, & \mathrm{E}\left[{X_M}^4\right] = 24.00, & \mathrm{E}\left[{X_M}^5\right] = 120.00 \end{array}$$

Calculating the *coefficient of variation* c_X we recognise that the above analysed phase-type example is more bursty than the negative exponential distribution ($c_M = 1$), but less bursty than the hyper-exponential process with $\lambda = (0.75, 1.5)$ and $\alpha = (0.5, 0.5)$.

$$c_X = \frac{\sqrt{Var(X)}}{\mathrm{E}[X]} = \frac{\sqrt{\mathrm{E}[X^2] - \mathrm{E}[X]^2}}{\mathrm{E}[X]} \quad \Rightarrow \quad c_{Ph_5} = 1.0256 \quad c_{H_2} = 1.1055$$

Due to the slightly too high mean value of the assumed phase-type process we cannot conclude on its characteristic by comparing the second raw moments directly. That the Ph_5 example is so

close was pure chance, its parameters were intuitively chosen toward achieving unit mean. The more phases we allow, the more degrees of freedom we have in selecting different parameters and the more difficult it becomes to define a process that meets certain characteristics. Commonly, H_2 and homogeneous *Cox* (mixed Erlang $E_k(\lambda, p)$) with equal $\lambda_i \; \forall_i$ and $p_i \; \forall_{i<k}$ are used [34]. These two are sufficient to perfectly match the first two moments for $c_X > 1$ and $c_X < 1$, respectively. How this is performed is outlined in section 2.3.

2.1.3 Markov Arrival Process – MAP

In 1979 M. F. Neuts introduced a *versatile Poisson process* [43] that later became known and referred to as the *Markov arrival process* (MAP). It is a quite general tool to model *point processes*, which still enables analytic treatment. Evidently, complex processes cause complex formulas. However, using the MAP to describe them, their structure is regular and can be handled by computers. Therefore, a MAP based definition of a process is well suited to be used with simulation studies, because it is a maximally flexible approach to process specification.

Please note that the term *arrival* in its name does not at all restrict this tool to arrival processes. For consistency in understanding we follow the community in the tradition to refer to this tool by its most widely known name, stressing that in general, and here very explicitly, a name may have no inherent meaning.

To explain the MAP and how a process can be defined using a MAP we reconsider the phase type generator (equation 2.13). We identify two types of transitions: internal ones that do not cause events, and exiting transitions that do cause events. In case of phase type renewal processes the generator is restarted after an event occurred, and the re-entry is independent of the event causing transition. If we include the event causing re-entry transitions in the generator phases mesh, we get the MAP definition. The transition rates for the MAP definition of a $Ph(\vec{\alpha}, T)$-distribution are

$$\begin{aligned} &\text{non-event causing} && r^*_{ij} = \pi_i t_{ij} = \pi_i p_{ij} \lambda_i \\ &\text{event causing} && r_{ij} = \pi_i (1 - \sum_j t_{ij}) \alpha_j = \pi_i (1 - \sum_j p_{ij}) \lambda_i \alpha_j \end{aligned}$$

where π_i is the phase's probability, being in steady state the time proportion that the generator is in phase i. Note that this is not equal to the holding time $\frac{1}{\lambda_i}$, it also depends on the likelihood with which the generator passes the individual phase.

We now relax the renewal condition that re-entries must be independent of the current event, and obtain the general MAP definition. Figure 2.11 shows an example for a 5-phase MAP definition.

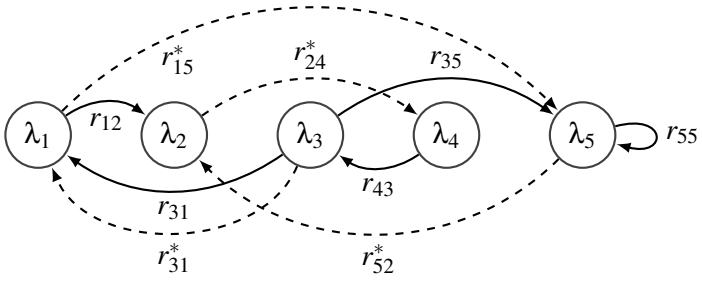


Figure 2.11: MAP definition example

In figure 2.11 the *dashed* transitions show non-event causing internal transitions, and the *solid* transitions show the event causing transitions. Note that we need self-loops to consider the case that the process remains in a generator phase after an event occurred (r_{55}). And there also may exist

ij-transitions that may or may not be related with an event (r_{31}, r^*_{31}). Latter demands to split the transitions into two sets, and thus we need two matrices to explicitly define the process's generator.

$$D_0 = \begin{bmatrix} -\lambda_1 & 0 & 0 & 0 & r^*_{15} \\ 0 & -\lambda_2 & 0 & r^*_{24} & 0 \\ r^*_{31} & 0 & -\lambda_3 & 0 & 0 \\ 0 & 0 & 0 & -\lambda_4 & 0 \\ 0 & r^*_{52} & 0 & 0 & -\lambda_5 \end{bmatrix} \qquad D_1 = \begin{bmatrix} 0 & r_{12} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ r_{31} & 0 & 0 & 0 & r_{35} \\ 0 & 0 & r_{43} & 0 & 0 \\ 0 & 0 & 0 & 0 & r_{55} \end{bmatrix} \tag{2.15}$$

In matrix D_0 we gather all non-event causing transitions r^*_{ij} and the phase's inverse holding times, comprising the negative diagonal entries $-\lambda_i$. In matrix D_1 we gather all event causing transitions r_{ij}, which evidently all need to be positive, if unequal zero. Both matrices are $k \times k$ matrices, where k is the number of generator phases, and the generator matrix Q results from simple matrix addition

$$Q_{MAP} = D_0 + D_1. \tag{2.16}$$

Compared to the phase-type generator there is no explicit exit option from the MAP generator, and thus we do not need an extra column to define the generator matrix. Still, to achieve a stationary generator every row in Q needs to sum to zero, $\sum_j q_{ij} = 0$.

The matrices D_0 and D_1 completely specify the MAP, and solving the equation system given by

$$\pi Q = 0, \quad \sum \pi_i = 1, \quad \pi_i \in [0 .. 1]$$

provides the invariant probability vector π of the continuous time Markov chain that underlies the process. The total arrival rate λ^* of the MAP is given by

$$\lambda^* = \pi D_1 1. \tag{2.17}$$

Returning to the phase-type generator we can now specify the Ph-process generator using the MAP approach:

$$D_0 = T \qquad\qquad D_1 = \vec{t_0} \times \vec{\alpha}. \tag{2.18}$$

For the Ph-example given in 2.1.2 equation 2.14 we get:

$$D_0 = \begin{bmatrix} -\lambda_1 & p_{12}\lambda_1 & p_{13}\lambda_1 & 0 & 0 \\ 0 & -\lambda_2 & p_{23}\lambda_2 & 0 & 0 \\ 0 & 0 & -\lambda_3 & p_{34}\lambda_3 & p_{35}\lambda_3 \\ 0 & 0 & 0 & -\lambda_4 & p_{45}\lambda_4 \\ p_{51}\lambda_5 & 0 & 0 & 0 & -\lambda_5 \end{bmatrix}, \quad D_1 = \begin{bmatrix} t_{0,1}\alpha_1 & t_{0,1}\alpha_2 & t_{0,1}\alpha_3 & t_{0,1}\alpha_4 & t_{0,1}\alpha_5 \\ t_{0,2}\alpha_1 & t_{0,2}\alpha_2 & t_{0,2}\alpha_3 & t_{0,2}\alpha_4 & t_{0,2}\alpha_5 \\ t_{0,3}\alpha_1 & t_{0,3}\alpha_2 & t_{0,3}\alpha_3 & t_{0,3}\alpha_4 & t_{0,3}\alpha_5 \\ t_{0,4}\alpha_1 & t_{0,4}\alpha_2 & t_{0,4}\alpha_3 & t_{0,4}\alpha_4 & t_{0,4}\alpha_5 \\ t_{0,5}\alpha_1 & t_{0,5}\alpha_2 & t_{0,5}\alpha_3 & t_{0,5}\alpha_4 & t_{0,5}\alpha_5 \end{bmatrix},$$

where we note that D_1 is completely filled. High filling is typical due to the per event renewal property of phase-type distributions: every possible exit state has to be connected with every possible entry state. The event causing transition rates r_{ij} of the corresponding MAP representation result as exit-rate $t_{0,i}$ times the entry probability α_j. In case we may exit from any state and may re-enter to any state, than D_1 has to be completely filled. For the numeric example in 2.1.2 we get

$$Q = \begin{bmatrix} -3 & 1.2 & 1.2 & 0 & 0 \\ 0 & -4 & 3.2 & 0 & 0 \\ 0 & 0 & -5 & 2.0 & 2.0 \\ 0 & 0 & 0 & -6 & 4.8 \\ 5.6 & 0 & 0 & 0 & -7 \end{bmatrix} + \begin{pmatrix} 0.6 \\ 0.8 \\ 1.0 \\ 1.2 \\ 1.4 \end{pmatrix} \times \begin{pmatrix} 0.10 \\ 0.15 \\ 0.20 \\ 0.25 \\ 0.30 \end{pmatrix} = \begin{bmatrix} -2.94 & 1.29 & 1.32 & 0.15 & 0.18 \\ 0.08 & -3.88 & 3.36 & 0.20 & 0.24 \\ 0.10 & 0.15 & -4.80 & 2.25 & 2.30 \\ 0.12 & 0.18 & 0.24 & -5.70 & 5.16 \\ 5.74 & 0.21 & 0.28 & 0.35 & -6.58 \end{bmatrix}$$

where evidently every row sums to zero.

For completeness we need to note that the MAP definition can be used to specify transient processes as well. These result if the topology of the phases contains transient parts and absorbing parts, being areas that cannot be reached again after a transition out of the area occurred, and areas that cannot be left any more once entered, respectively. Considering such expresses the utility of the MAP tool. However, to model distribution functions we need steady processes and thus we consider henceforth structures that neglect transient areas, and exclude absorbing phases. Transient areas have no influence on the steady state behaviour, and absorbing phases would stop the generator prior reaching the steady state. Each absorbing area defines the steady state of a stable generator. If more than one absorbing area is possible we need to consider them independently. They actually are independent, they do not influence each other, and therefore, we can join the individual results by considering the likelihood to enter either absorbing area as weighting factor.

2.1.4 Composed MAP processes and the extension to batch arrivals

Having outlined the versatile MAP tool we briefly introduce two process examples very useful for practical problems, being the *Interrupted Poisson Process* (IPP) and the more general *Markov Modulated Poisson Process* (MMPP). We use these special cases to highlight some features of the MAP as well as potential inconsistencies that result from different definition approaches. Next we outline *composed processes*, where the MAP approach is used to join phase diagrams of particular processes in order to design processes that incorporate different characteristics.

Concluding the MAP, we sketch its extension to consider batched arrivals. Actually this is an extension of the MAP approach. However, it is quite straightforward, and therefore we present the *Batch Markovian Arrival Process* (BMAP) here in line.

Interrupted Poisson process – IPP

Consider the system shown on the left side of figure 2.12. The switch causes negative exponentially distributed events only during randomly occurring time intervals, where the duration of the intervals during which events may or may not occur are themselves negative exponentially distributed.

Figure 2.12: Interrupted Poisson process (IPP) and its definition as MAP

If the on-off intervals are given by the switching rates ε_{on} and ε_{off}, than the average on-duration $\tau_{on} = \frac{1}{\varepsilon_{off}}$ and the average off-duration $\tau_{off} = \frac{1}{\varepsilon_{on}}$, because the rates define how fast the current state is left toward the state that is shown as index with the according ε. We can thereby immediately calculate the average event rate

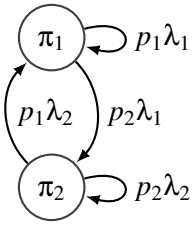
$$\lambda^* = \lambda \frac{\tau_{on}}{\tau_{on} + \tau_{off}} = \lambda \frac{\frac{1}{\varepsilon_{off}}}{\frac{1}{\varepsilon_{off}} + \frac{1}{\varepsilon_{on}}} = \lambda \frac{\varepsilon_{on}}{\varepsilon_{on} + \varepsilon_{off}},$$

where this on-off mismatch between τ- and ε-indices is nicely hidden.

Using the MAP approach we get

$$Q_{IPP} = D_0 + D_1 = \begin{bmatrix} -\varepsilon_{on} & \varepsilon_{on} \\ \varepsilon_{off} & -(\varepsilon_{off} + \lambda) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} -\varepsilon_{on} & \varepsilon_{on} \\ \varepsilon_{off} & -\varepsilon_{off} \end{bmatrix}. \tag{2.19}$$

Quite interestingly equals the IPP stochastically a two-phase hyper-exponential process. To prove this, we define the H_2 process as MAP (figure 2.13) and compare the two MAP structures.



$$D_0 = \begin{bmatrix} -\lambda_1 & 0 \\ 0 & -\lambda_2 \end{bmatrix}, \qquad D_1 = \begin{bmatrix} p_1\lambda_1 & p_2\lambda_1 \\ p_1\lambda_2 & p_2\lambda_2 \end{bmatrix}$$

$$Q = \begin{bmatrix} -(1-p_1)\lambda_1 & p_2\lambda_1 \\ p_1\lambda_2 & -(1-p_2)\lambda_2 \end{bmatrix} = \begin{bmatrix} -p_2\lambda_1 & p_2\lambda_1 \\ p_1\lambda_2 & -p_1\lambda_2 \end{bmatrix}$$

Figure 2.13: MAP representation of a H_2 generator

For $\varepsilon_{on} = p_2\lambda_1$ and $\varepsilon_{off} = p_1\lambda_2$ the two generator matrices become identical, which proves that the two processes are stochastically equivalent.

Markov modulated Poisson process – MMPP

Extending the above introduced Poisson process to provide multiple phases during which events occur at different rates λ_i we get the so called *Markov modulated Poisson process* (MMPP). The most commonly used model is the two phases MMPP where the event generator toggles negative exponentially distributed between two event generation rates. This is very similar the the IPP example above, and using the MAP approach we get

$$Q_{MMPP_2} = D_0 + D_1 = \begin{bmatrix} -(\varepsilon_{12}+\lambda_1) & \varepsilon_{12} \\ \varepsilon_{21} & -(\varepsilon_{21}+\lambda_2) \end{bmatrix} + \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} -\varepsilon_{12} & \varepsilon_{12} \\ \varepsilon_{21} & -\varepsilon_{21} \end{bmatrix}, \qquad (2.20)$$

which becomes the IPP if we set $\lambda_1 = 0$.

If we consider more rates, meaning phases, different switch-over policies can be define. Consider the system shown on the left side of figure 2.14, where the different rates are cycled through in round-robin order with the individual switch-over rates $\varepsilon_{12}, \varepsilon_{23}, \varepsilon_{31}$.

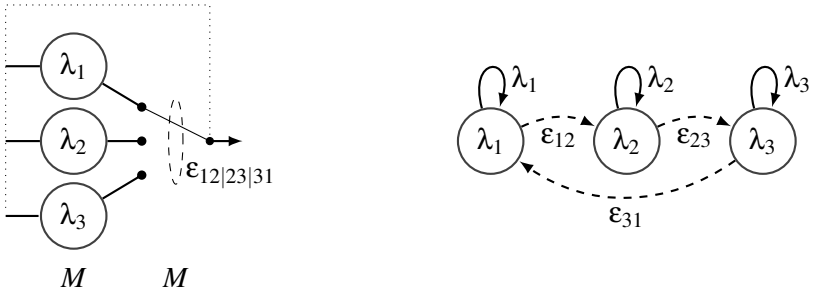


Figure 2.14: Markov modulated Poisson process (MMPP) and its definition as MAP

The duration of the time intervals during which events are generated at rate λ_i are still negative exponentially distributed, and their average durations are given by $\tau_1 = \frac{1}{\varepsilon_{12}}$, $\tau_2 = \frac{1}{\varepsilon_{23}}$, $\tau_3 = \frac{1}{\varepsilon_{31}}$. We can thereby immediately calculate the average event rate of the generator

$$\lambda^* = \frac{\tau_1\lambda_1 + \tau_2\lambda_2 + \tau_3\lambda_3}{\tau_1+\tau_2+\tau_3} = \frac{\frac{\lambda_1}{\varepsilon_{12}} + \frac{\lambda_2}{\varepsilon_{23}} + \frac{\lambda_3}{\varepsilon_{31}}}{\frac{1}{\varepsilon_{12}} + \frac{1}{\varepsilon_{23}} + \frac{1}{\varepsilon_{31}}} = \frac{\varepsilon_{23}\varepsilon_{31}\lambda_1 + \varepsilon_{12}\varepsilon_{31}\lambda_2 + \varepsilon_{12}\varepsilon_{23}\lambda_3}{\varepsilon_{12}\varepsilon_{23} + \varepsilon_{12}\varepsilon_{31} + \varepsilon_{23}\varepsilon_{31}}.$$

Using the MAP approach we get

$$Q = \begin{bmatrix} -(\varepsilon_{12}+\lambda_1) & \varepsilon_{12} & 0 \\ 0 & -(\varepsilon_{23}+\lambda_2) & \varepsilon_{23} \\ \varepsilon_{31} & 0 & -(\varepsilon_{31}+\lambda_3) \end{bmatrix} + \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} = \begin{bmatrix} -\varepsilon_{12} & \varepsilon_{12} & 0 \\ 0 & -\varepsilon_{23} & \varepsilon_{23} \\ \varepsilon_{31} & 0 & -\varepsilon_{31} \end{bmatrix}. \qquad (2.21)$$

If we now define a different switch-over policy we get a different generator. Consider the system shown on the left side of figure 2.15, where the different generation rates are switched among neighbouring phases only, with the switch-over rates ε_{12}, ε_{21}, ε_{23}, ε_{32}.

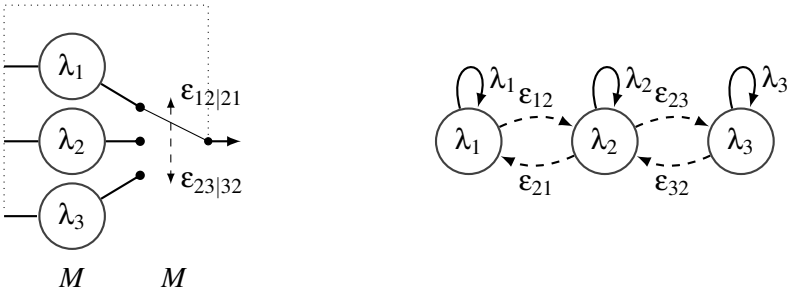


Figure 2.15: Alternate Markov modulated Poisson process (MMPP) and its definition as MAP

The duration of the time intervals during which events are generated at rate λ_i are still negative exponentially distributed, and their average durations are given by $\tau_1 = \frac{1}{\varepsilon_{12}}$, $\tau_2 = \frac{1}{\varepsilon_{21}+\varepsilon_{23}}$, $\tau_3 = \frac{1}{\varepsilon_{32}}$. Here we cannot immediately calculate the average event rate of the generator, because the number of phase visits is not equal. Intuitively we might think that the middle state would be visited twice as often; however, if we set one outgoing ε to zero, we recognise that the phase probabilities depend on the actual switch-over rates. We need to solve the equilibrium equations for the phase probabilities π_i first.

$$\begin{array}{rcrcrcl} -\pi_1\varepsilon_{12} & + & \pi_2\varepsilon_{21} & + & 0 & = & 0 \\ \pi_1\varepsilon_{12} & - & \pi_2(\varepsilon_{21}+\varepsilon_{23}) & + & \pi_3\varepsilon_{32} & = & 0 \\ 0 & + & \pi_2\varepsilon_{23} & - & \pi_3\varepsilon_{32} & = & 0 \\ \hline \pi_1 & + & \pi_2 & + & \pi_3 & = & 1 \end{array} \quad \Rightarrow \quad \pi = \begin{pmatrix} \frac{\varepsilon_{21}\varepsilon_{32}}{\varepsilon_{12}\varepsilon_{23}+\varepsilon_{12}\varepsilon_{32}+\varepsilon_{21}\varepsilon_{32}} \\ \frac{\varepsilon_{12}\varepsilon_{32}}{\varepsilon_{12}\varepsilon_{23}+\varepsilon_{12}\varepsilon_{32}+\varepsilon_{21}\varepsilon_{32}} \\ \frac{\varepsilon_{12}\varepsilon_{23}}{\varepsilon_{12}\varepsilon_{23}+\varepsilon_{12}\varepsilon_{32}+\varepsilon_{21}\varepsilon_{32}} \end{pmatrix} \tag{2.22}$$

Having the phase probabilities, the average event generation rate can be immediately calculated by

$$\lambda^* = \pi_1\lambda_1 + \pi_2\lambda_2 + \pi_3\lambda_3 = \frac{\varepsilon_{21}\varepsilon_{32}\lambda_1 + \varepsilon_{12}\varepsilon_{32}\lambda_2 + \varepsilon_{12}\varepsilon_{23}\lambda_3}{\varepsilon_{12}\varepsilon_{23} + \varepsilon_{12}\varepsilon_{32} + \varepsilon_{21}\varepsilon_{32}}.$$

Using the MAP approach we get

$$Q = \begin{bmatrix} -(\varepsilon_{12}+\lambda_1) & \varepsilon_{12} & 0 \\ \varepsilon_{21} & -(\varepsilon_{21}+\varepsilon_{23}+\lambda_2) & \varepsilon_{23} \\ 0 & \varepsilon_{32} & -(\varepsilon_{32}+\lambda_3) \end{bmatrix} + \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} = \begin{bmatrix} -\varepsilon_{12} & \varepsilon_{12} & 0 \\ \varepsilon_{21} & -(\varepsilon_{21}+\varepsilon_{23}) & \varepsilon_{23} \\ 0 & \varepsilon_{32} & -\varepsilon_{32} \end{bmatrix} \tag{2.23}$$

and recognise that the generator matrix Q states the equilibrium equations required to calculate the phase probabilities vector $\vec{\pi}$.

$$Q\pi = 0 \qquad \Rightarrow \qquad \pi = Q^{-1}0 \tag{2.24}$$

As common with equilibrium equations, the matrix is singular and the above stated equation cannot be solved because the right equation corresponds to a division of Q by the zero vector $\vec{0}$. We have to replace one row by the side condition that assures that the generator is in some phase at any time, being $\sum_i \pi_i = 1$. See equation 2.22, where this step has been applied explicitly.

The simple examples shown above indicate how useful the MMPP approach is. Whenever it is possible to identify negative exponentially distributed phases with stable mean event rates, and we know how long these phases in average last until a next phase is entered, and finally also know the transition probabilities among phases, being the switch-over policy, than we can immediately define the MMPP that models the process in total.

Composing of a MAP processes by joining different processes

The MMPP presented above enables us to design processes composed of individual event generating Markov phases. Using the MAP approach, we can extend this to the composition of processes by combining different basic processes, which themselves are compositions of not necessarily event generating Markov phases. For example, an Erlang distributed interruption period can be integrated within a hyper-exponential generator, as shown in figure 2.16.

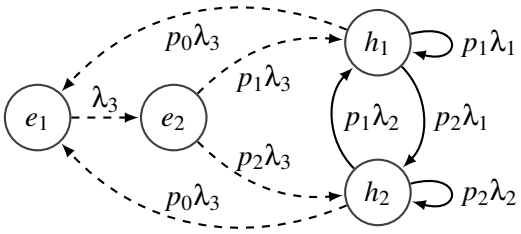


Figure 2.16: H_2 generator with E_3 distributed interruptions

$$D_0 = \begin{bmatrix} -\lambda_3 & \lambda_3 & 0 & 0 \\ 0 & -\lambda_3 & p_1\lambda_3 & p_2\lambda_3 \\ p_0\lambda_3 & 0 & -(\lambda_1+p_0\lambda_3) & 0 \\ p_0\lambda_3 & 0 & 0 & -(\lambda_2+p_0\lambda_3) \end{bmatrix}, \quad D_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & p_1\lambda_1 & p_2\lambda_1 \\ 0 & 0 & p_1\lambda_2 & p_2\lambda_2 \end{bmatrix}$$

$$Q = \begin{bmatrix} -\lambda_3 & \lambda_3 & 0 & 0 \\ 0 & -\lambda_3 & p_1\lambda_3 & p_2\lambda_3 \\ p_0\lambda_3 & 0 & p_2\lambda_1-p_0\lambda_3 & p_2\lambda_1 \\ p_0\lambda_3 & 0 & p_1\lambda_2 & p_1\lambda_2-p_0\lambda_3 \end{bmatrix}$$

In this example we use p_0 for the interruption probability and p_1, p_2 for the conditional generation rate probabilities, given that the generation is currently not interrupted. Thus, $p_1 + p_2 = 1$ and $p_0 \in [0\,..\,1]$ can be independently chosen. Still, the sum of all phase probabilities π_i needs to be one ($\sum\pi_i$=1). We can derive them step by step; first we solve the equilibrium equations for π_e and π_h, the probability to be interrupted or not, respectively

$$\underbrace{\pi_{e_2}}_{=\pi_e/2}\underbrace{(p_1+p_2)}_{=1}\lambda_3 = \underbrace{(\pi_{h_1}+\pi_{h_2})}_{=\pi_h}p_0\lambda_3 \quad \Rightarrow \quad \frac{\pi_e}{2} = \pi_h p_0 \underset{\pi_e+\pi_h=1}{\Rightarrow} \pi_e = \frac{2p_0}{1+2p_0}, \quad \pi_h = \frac{1}{1+2p_0},$$

and secondly, we split these according to the rules of the internal processes, to get

$$\pi_{e_1} = \pi_{e_2} = \frac{p_0}{1+2p_0}, \qquad \pi_{h_1} = \frac{p_1}{1+2p_0}, \quad \pi_{h_2} = \frac{p_2}{1+2p_0},$$

entirely expressed by the p_i only, which corresponds with the known rate independence of the basic processes ($\pi_i(E) = \frac{1}{k}$, $\pi_i(H) = p_i$).

The here applied solving approach is an example for the *de-compositioning* of a complex process into encapsulated less complex processes. Here it is evident, because we composed the process by joining less complex ones. The application of this method where it is not so apparent, is discussed in detail where it has been used to study particular systems throughout upcoming chapters.

The example roughly indicates the multitude of combination options. Whenever we expect certain distributions during different time intervals, we can try to model the process by combining their MAP representation into a bigger MAP. However, if we need to fit the model parameters to measured statistical figures, than the least complex, meaning the model with the smallest number

of phases, is the best choice. The models typically provide more parameters than there are figures to match them to. Consequently we need to choose several and fit the remaining. However we do this, the designed model's behaviour will fit the figures, but it may behave strange in other aspects, depending on the choices we made. Statistical matching yields at least similar processes, but not necessarily identical ones.

Batch Markovian arrival process – BMAP

A restriction of all the above models is that at a certain time only a single event may occur. For time continuous systems this can always be assumed. However, if events occur in groups, meaning that many events occur in a very short time interval followed by a comparably long time interval during which no events occur, and, in addition, we do not need to care for the order of events within these groups and the group size is independent and identically distributed over group arrivals, than, it is straightforward to model the group arrival distribution and the group size distribution by two independent processes. The groups we call *batches*, and we now consider events that correspond to a batch-arrival together with the size of the arrived batch, being the number of events the current batch is composed of.

If we look at the definition of the MAP (equation 2.16), we see that it is composed of two matrices, D_0 holding the transitions that do not relate to event occurrences, and D_1 holding the transitions that are related with the occurrence of an event. If we extend this by matrices D_n holding transitions related to the occurrence of n events, meaning related to the event that an events-batch of size n occurs, than we get the definition of the *Batch Markovian Arrival Process* (BMAP).

$$Q_{BMAP} = D_0 + D_1 + D_2 + D_3 + \ \ldots \ = \sum_{i=0}^{\infty} D_i \tag{2.25}$$

The matrices D_i completely specify the BMAP, and solving the equation system given by

$$\pi Q = 0, \quad \sum \pi_i = 1, \quad \pi_i \in [0 \,..\, 1]$$

provides the invariant probability vector π of the continuous time Markov chain that underlies the process. The batch arrival rate $\lambda^{(\text{batch})}$ is given by

$$\lambda^{(\text{batch})} = -\pi D_0 1 \tag{2.26}$$

and the total event rate λ^* of the BMAP is given by

$$\lambda^* = \pi \left(\sum_{i=1}^{\infty} i D_i \right) 1. \tag{2.27}$$

As an example for the definition of a BMAP let us consider phase-type distributed batch occurrences, given by $Ph(\vec{\alpha}, T)$, with Poisson distributed batch size X_{batch}, where $p_i = \text{P}[x{=}i]$ and $x = \text{E}[X_{\text{batch}}] = \sum i\, p_i$ are given. The sub-matrices specifying this BMAP are

$$D_0 = T, \quad D_i = p_i \left(\vec{t_0} \times \vec{\alpha} \right) \qquad \text{where} \quad \vec{t_0} = -T1 \quad \text{and} \quad p_i = \frac{x^i}{i!} e^{-x}$$

such that

$$Q = T + \sum_{i=1}^{\infty} p_i \left(\vec{t_0} \times \vec{\alpha} \right)$$

we see that this is not very handy due to the infinite sum. However, if the batch size X_{batch} is upper bounded, which likely is the case with real systems, the method becomes practically utile.

2.2 Non-Markovian processes

The *Markovian* processes discussed in section 2.1 can in theory be used to model any characteristic. However, some specific processes with special properties require an infinite number of phases. Thus, the complexity to sufficiently approximate these may be intractable. Here we discuss some well known processes that can be used instead.

Two extremes sometimes required to model real world processes are the *deterministic* process (D) and the *uniform* distribution (U). Extending the double bounded uniform distribution we present later on the *Beta* distribution ($\mathcal{B}$), which can be used to model a wide range of bounded distributions, including distributions with poles (peaks) at boundaries. Prior that, we introduce the family of *power-law* distributions (P), in particular three related *Pareto* distribution types, because of their *heavy tails*. For particular parameters these distributions can model 'chaotic' behaviour. Note, commonly we assume a stable and finite mean, which may not be in line with other definitions of chaotic systems. However, in a finite world random variables with an infinite variance may appear as if having unstable mean, because infinite variance causes an estimation problem: we need infinitely many samples to reliably estimate the mean.

Some non-Markovian distributions and their abbreviations used hereinafter

- D *Deterministic*: constant distributed events – infinite memory
- U *Uniform*: flat distributed events – equally likely bounded area $[a\,..\,b]$
- P *Pareto*: power-law distributed events – heavy tailed, infinite variance possible
- $\mathcal{B}$ *Beta*: double bounded event distributions – may peak at boundaries
- GI *general independent*: events caused by any renewal process (commonly i.i.d. arrivals)
- G *general*: any event distribution, potentially correlated and state dependent

2.2.1 Deterministic distribution

For *deterministic* distributed events the entire future is perfectly known once two occurrences have been monitored, because the variance is zero. Strictly speaking one may argument that the outcomes of a deterministic distributed process are not distributed at all, because they are perfectly correlated. However, in case of asynchronouse systems and time continuouse approaches, the central limit theorem is applicable for deterministic processes as well, and thus, $D(\mu)$ does represent an includeable extreme case. Only for synchronised systems and transient analysis this is not possible, because for these the initial conditions influence on the result does not decrease over time.

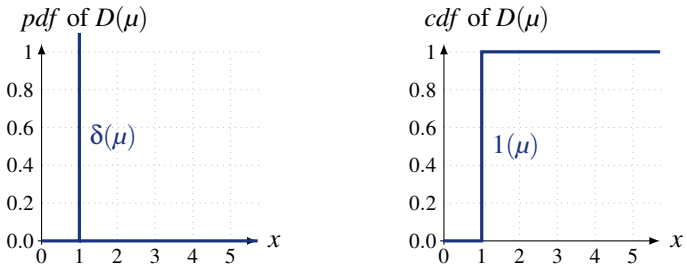


Figure 2.17: *pdf* and *cdf* of the deterministic distribution $D(\mu)$ for μ=1

Figure 2.17 shows on the left the probability density function (*pdf*) of $D(\mu)$, being the *Dirac delta function* $\delta(x-\mu)$ with intensity one positioned at the (mean) value $x=\mu$, and on the right its cumulative distribution function (*cdf*), being the *Heaviside unit-step function* $1(x-\mu)$, where the unit step occurs at $x=\mu$.

Deterministic distribution

$$\textit{pdf}: \qquad f_D(x) = \delta(x-\mu) = \begin{cases} \infty, & x = \mu \\ 0, & x \neq \mu \end{cases} \tag{2.28}$$

$$\textit{cdf}: \qquad F_D(x) = 1(x-\mu) = \begin{cases} 0, & x < \mu \\ 1, & x \geq \mu \end{cases} \tag{2.29}$$

$$\textit{mean}: \qquad \mathrm{E}[X_D] = \mu \tag{2.30}$$

$$\textit{variance}: \qquad Var(X_D) = 0 \tag{2.31}$$

$$\textit{coefficient of variation}: \qquad c_X = 0 \tag{2.32}$$

These *generalised* functions are very useful for analytic analysis, because they have Laplace transforms, $\delta(x-\mu) \hookrightarrow e^{-s\mu}$ and $1(x-\mu) \hookrightarrow \frac{e^{-s\mu}}{s}$. They are commonly used as *indicator functions* when it is necessary to integrate conditions into functions. The *delta function* can be used to integrate an *equality* condition, and the *unit-step* to integrate an *inequality* based condition. Evidently is $1(x-\mu) = \int_{-\infty}^{x} \delta(x-\mu)\,\mathrm{d}x$ and vice versa $\delta(x-\mu) = \frac{\mathrm{d}\,1(x-\mu)}{\mathrm{d}x}$.

Concerning distributions, the *Dirac delta function* is commonly defined as the extreme case of the *normal distribution* $N(\mu,\sigma^2)$, because $D(\mu) \equiv N(\mu,0)$, the limit distribution for $\sigma^2 \to 0$. For here it is worth to note that according to the theory of great numbers the deterministic distribution may be defined as the limiting case of the Erlang distribution as well, $D(\mu) \equiv E_k(k\lambda)$ for $k \to \infty$.

$$\delta(x-\mu) = \lim_{\sigma^2 \to 0} \left\{ f_{N(\mu,\sigma^2)} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right\} \qquad \delta(x-\frac{1}{\lambda}) = \lim_{k \to \infty} \left\{ f_{E_k(k\lambda)} = k\lambda \frac{(k\lambda x)^{k-1}}{(k-1)!} e^{-k\lambda x} \right\}$$

Note, when convenient we use μ to represent mean values (durations) and λ to represent mean rates, and that the former resemble the inverse of the latter ($\mu \bowtie \frac{1}{\lambda}$). The mathematical prove of the equations above is left to the experts. For now and further on, it is sufficient to recognise the equality graphically, remembering that the *pdf* or *cdf* specify any distribution definitely. Obviously causes

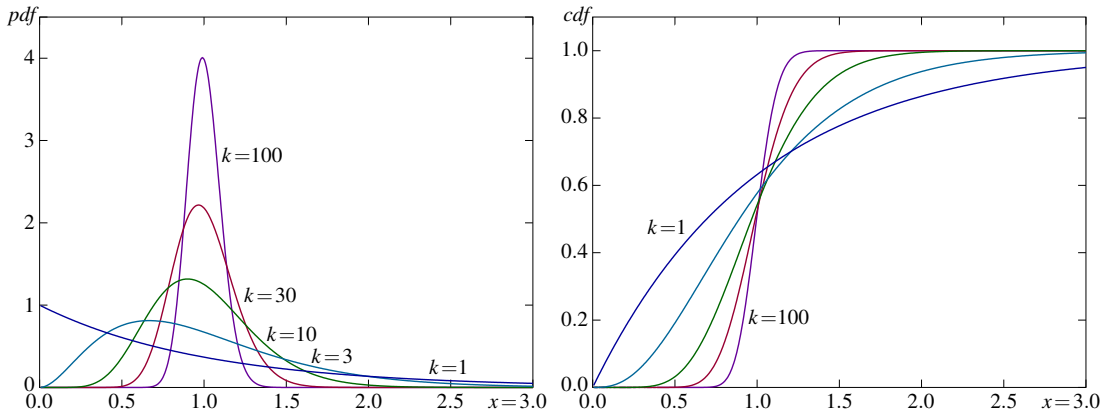


Figure 2.18: For $k \to \infty$ approximates $E_k(\lambda)$ the deterministic $D(\mu)$ if we set $\lambda = k\mu$

$\lambda = k\mu$ a practical problem because it forces $\lambda \to \infty$. In particular, the $(k-1)!$ in the denominator of the distribution function causes numeric problems for large k. The example presented in figure 2.18 for $\mu=1$ shows that for 100 phases E_k the variance, being the width of the $pdf(x)$, is far from negligible, and E_{300} could no more be calculated by *Octave* [29].

2.2.2 Uniform distribution

If all possible outcomes of a random process are equally likely, than they are *uniformly* distributed. This is only possible if the sample space is restricted, meaning *bounded* to a certain interval on the x-axis. If the range would not be restricted to a defined interval the probability density would become zero for all values and a mean value would not exist. Consequently, the *uniform distribution* stresses that the outcomes of a uniform distributed event generation process are definitely restricted to a certain interval. The uniform distribution is commonly specified by the two bounds as $U(a,b)$, and the mean value results as $\mu = \frac{a+b}{2}$. Alternatively we might specify it also by $U(a,\mu)$, the lower bound and the mean value, because the upper bound results as $b = 2\mu - a$. This form is convenient for uniform event generation process with 0 being the lower bound, which for any causal event distribution always holds. For this case we get the interval $[0 < u(i) < 2\mu]$, and stipulate this interval if only a single parameter is given $U(\mu) \equiv U(0,2\mu)$. Anyhow, we need to note that in practice the bounds determine the mean value of the distribution, not vice versa.

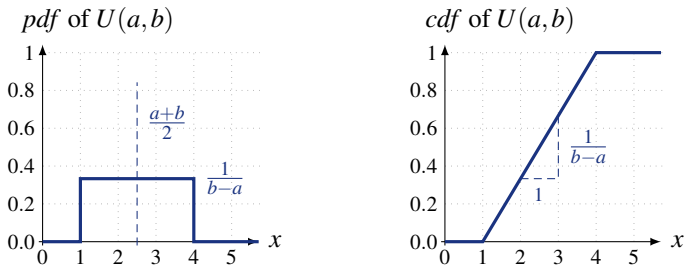


Figure 2.19: *pdf* and *cdf* of the uniform distribution $U(a,b)$ for a=1, b=4 $(\rightarrow \mu = \frac{a+b}{2} = 2.5)$

Figure 2.19 shows on the left the probability density function (*pdf*) of $U(a,b)$, being a *bandpass filter* function with bounds a and b and the filter gain $\frac{1}{b-a}$, and on the right its cumulative distribution function (*cdf*), being a *slope* function, where between a and b the value rises linearly from 0 to 1 with the gradient $\frac{1}{b-a}$.

Uniform distribution

pdf:
$$f_U(x) = \begin{cases} 0, & x < a \\ \frac{1}{b-a}, & a \le x < b \\ 0, & x \ge b \end{cases} \tag{2.33}$$

cdf:
$$F_U(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \le x < b \\ 1, & x \ge b \end{cases} \tag{2.34}$$

mean:
$$\mathrm{E}[X_U] = \frac{a+b}{2} \tag{2.35}$$

variance:
$$Var(X_U) = \frac{(b-a)^2}{12} \tag{2.36}$$

coefficient of variation:
$$c_X = \frac{1}{\sqrt{3}} \frac{b-a}{a+b} \tag{2.37}$$

The uniform distribution $U(a,b)$ is very useful to model bounded processes. Its maximal *coefficient of variation* occurs for $a = 0$ as $c_{X\,max} = \frac{1}{\sqrt{3}} < 1$, which expresses that the uniform distribution is less bursty than the negative exponential distribution.

A special role plays $U(0,1)$, because it is commonly provided by random number generators and is widely used to generate somehow distributed random numbers, using the inversion method

to convert uniformly distributed numbers in $[0\,..\,1]$ into the desired distribution $X = F_X^{-1}(U(0,1))$, which not necessarily is bounded. $F_X^{-1}(y)$ is the inverse of the *cdf*, which is also called the *quantile function* because it returns the quantiles for y. Graphically this method relates uniformly distributed y-axis values of the *cdf* with the corresponding x-axis values, which than are distributed according to F_X. For example can negative exponentially distributed numbers be generated via the natural logarithm of a uniform distributed variable $u = X_{U(0,1)}$ by

$$X_M(\lambda) = \frac{-ln(1-u)}{\lambda} = \frac{-ln(u)}{\lambda}, \tag{2.38}$$

where we use the uniformity property that $\mathrm{P}[y{=}1{-}u] = \mathrm{P}[y{=}u]$. Power law distributed random variables can be generated via the inverse root of a uniformly distributed variable, in particular can *Lomax distributed* numbers be generated by

$$X_{P_L}(\mu,\alpha) = \mu\left(u^{-\frac{1}{\alpha}} - 1\right) = \mu\,\frac{1-\sqrt[\alpha]{u}}{\sqrt[\alpha]{u}}, \tag{2.39}$$

where $\mathrm{E}[X_{P_L}] = \frac{\mu}{\alpha-1}$ is defined $\forall_{\alpha>1}$ only.

2.2.3 Heavy tailed distributions

The above presented examples of non-Markovian distributions are all less bursty than the negative exponential distribution. In certain cases, particularly if events are positively correlated, we need so-called *heavy-tailed* distributions to model their behaviour. Heavy-tailed refers to a significant *pdf* part above the 90%-percentile. The impact (*weight*) of these not very likely events on the mean and the variance is significant, because of the diminishing gradient of the *pdf* for increasing x.

The quite simple *power function* does have this property. Even though the function itself is simple, analytic derivations are out of the scope here, because its Laplace transform is not very handsome. Quite interestingly, for certain parameter ranges these distribution functions model *chaotic* processes behaviour; where 'chaotic' is experienced as infinite variance $Var(X){=}\infty$, simply.

Pareto distributions

The *Pareto distribution* follows a power law (cx^{-a}), in contrast to the Markovian distributions, which are based on the exponential function (ca^{-x}). For a certain parameter range Pareto yields a well defined distribution with finite mean and infinite variance. This can be used to model heavy tailed processes with positively correlated events.

The pdf $f_X(x) = cx^{-a}$ for $x \geq min$, and equal 0 for $x < min$, with constant c chosen to get $\int_{-\infty}^{\infty} f_X(x) = 1$. This constraint is implicitly fulfilled if we substitute $a = \alpha + 1$ and $c = \alpha\beta^{\alpha}$, where α defines the *shape* and β is the lower *boundary* ($\beta = min$). Together they are the distribution's parameters to choose. In this basic form, the *Pareto type-1 distribution*, it can be used to model processes where a lower bound exists and the probabilities decay according to a power law above this limit.

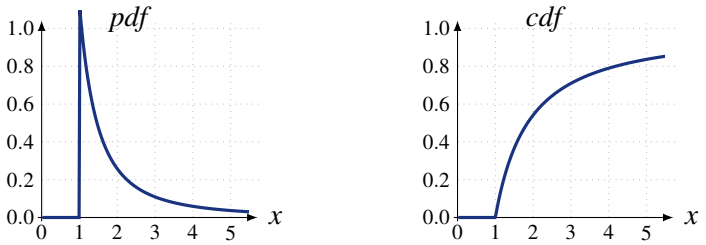


Figure 2.20: *pdf* and *cdf* of the *Pareto type-1 distribution* (α=1.125, β=1)

Figure 2.20 shows on the left the probability density function (*pdf*) of $P(\alpha,\beta)$ and on the right its cumulative distribution function (*cdf*). Note that for the chosen parameters the *cdf* does not come close to 1 quickly. This indicates that the tail of the distribution, being the *pdf* part above $x = F_P^{-1}(0.9)$, is heavy, meaning that its contribution to the characteristic is significant.

Pareto distribution – Pareto type-1

pdf:
$$f_P(x) = \begin{cases} \alpha\beta^\alpha x^{-(\alpha+1)} & x \geq \beta \\ 0 & x < \beta \end{cases} \tag{2.40}$$

cdf:
$$F_P(x) = \begin{cases} 1-\beta^\alpha x^{-\alpha} & x \geq \beta \\ 0 & x < \beta \end{cases} \tag{2.41}$$

mean:
$$\mathrm{E}[X_P] = \begin{cases} \frac{\alpha\beta}{\alpha-1} & \alpha > 1 \\ \infty & \alpha \leq 1 \end{cases} \tag{2.42}$$

variance:
$$Var(X_P) = \begin{cases} \frac{\alpha\beta^2}{(\alpha-2)(\alpha-1)^2} & \alpha > 2 \\ \infty & \alpha \leq 2 \end{cases} \tag{2.43}$$

coefficient of variation:
$$c_X = \begin{cases} \sqrt{\frac{1}{\alpha(\alpha-2)}} & \alpha > 2 \\ \infty & \alpha \leq 2 \end{cases} \tag{2.44}$$

For stability $\alpha>1$ is required, else the mean becomes infinite. However, the *pdf* and *cdf* can be calculated for any positive α and β. For $[1<\alpha\leq 2]$ the *Pareto distribution* is characterised by a finite mean value and infinite variance. This is the α-region of especial interest, commonly used to model really heavy tailed characteristics. For $\mu \to \beta$ the distribution approximates the Dirac delta function $\delta(\mu)$.

To generate Pareto distributed numbers by *cdf* inversion we might use

$$X_P = \frac{\beta}{\sqrt[\alpha]{1-u}} = \beta u^{-\frac{1}{\alpha}},$$

where $u = X_{U(0,1)}$ are uniformly distributed random numbers in $[0\,..\,1]$, and $1-u$ can be replaced by u due to uniformity. However, to generate a Pareto distributed random number with a certain mean value μ we need to either substitute $\alpha = \frac{\mu}{\mu-\beta}$ or $\beta = \mu\left(\frac{\alpha-1}{\alpha}\right)$, in order to get a distribution with the intended mean $\mu = \frac{\alpha\beta}{\alpha-1}$.

$$X_P(\mu) = \beta u^{-\frac{\mu-\beta}{\mu}} = \mu\left(\frac{\alpha-1}{\alpha}\right) u^{-\frac{1}{\alpha}} \tag{2.45}$$

Adopting α causes a rate dependent distribution shape, comparable to the rate dependent decline of the negative exponential distribution. The second option provides a generation with constant power law exponent α. This is compliant with the characterisation by the *coefficient of variation* or the *Hurst parameter*, which both depend on α only. A drawback is the mean dependent lower bound.

To compare Pareto with other distributions and to validate the unsteadiness within the unsteady α-region where $c_{X_P} = \infty$, the Hurst parameter can be used.

Hurst parameter:
$$H(X_P) = \frac{3-\alpha}{2}$$

For the region $[1<\alpha\leq 2]$ we get $H = [1\,..\,0.5]$, which indicates that the increments of an unsteady Pareto process are positively correlated.

This Pareto variant, strictly called Pareto type-1, is well suited to model packet traffic. The lower bound β is given by the minimum packet size, covering at least the header-size. The variable payload

length is than modelled by the shape parameter α. This is straight forward, and evidently works to model processing-, transmission-, and holding-times in general. Similarly, arrival processes with a minimum offset in between subsequent arrival events may be modelled, though such seem to be not very common. In that case it does make sense to assume a mean determined lower bound and a persistent shape for the distribution decline above the bound.

Shifted Pareto distribution – the Lomax distribution

To get a power law distribution with no lower bound β we substitute x by $x+\sigma$ and get the *Lomax distribution*, also known as *shifted Pareto* or *Pareto2*. We indicate it by P_L to express that it belongs to the power law distributions. It resembles a standard Pareto distribution shifted along the x-axis so to start at $x = 0$. The Lomax distribution P_L is again characterised by two parameters, the *shape* parameter α, which we already know, and a new *scale* parameter σ, instead of the lower boundary.

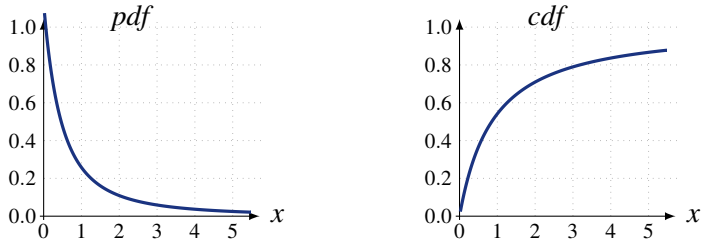


Figure 2.21: *pdf* and *cdf* of the *Lomax distribution* (α=1.125, σ=1)

Figure 2.21 shows on the left the probability density function (*pdf*) of $P_L(\alpha,\sigma)$ and on the right its cumulative distribution function (*cdf*). Again, for the chosen parameters the *cdf* does not come close to 1 quickly, indicating that the tail of the distribution (*pdf* part above $x = F_P^{-1}(0.9)$) is heavy and thus, significant for the process's characteristic.

Lomax distribution (P_L)

pdf:
$$f_{P_L}(x) = \alpha\sigma^{\alpha}(x+\sigma)^{-(\alpha+1)} \tag{2.46}$$

cdf:
$$F_{P_L}(x) = 1-\sigma^{\alpha}(x+\sigma)^{-\alpha} \tag{2.47}$$

mean:
$$\mathrm{E}[X_{P_L}] = \begin{cases} \frac{\sigma}{\alpha-1} & \alpha > 1 \\ \textit{undefined} & \alpha \leq 1 \end{cases} \tag{2.48}$$

variance:
$$Var(X_{P_L}) = \begin{cases} \frac{\alpha^2\sigma}{(\alpha-2)(\alpha-1)^2} & \alpha > 2 \\ \infty & 1 < \alpha \leq 2 \\ \textit{undefined} & \alpha \leq 1 \end{cases} \tag{2.49}$$

coefficient of variation:
$$c_X = \begin{cases} \sqrt{\frac{\alpha^2}{\sigma(\alpha-2)}} & \alpha > 2 \\ \infty & 1 < \alpha \leq 2 \\ \textit{undefined} & \alpha \leq 1 \end{cases} \tag{2.50}$$

For stability $\alpha>1$ is required, else the mean becomes infinite. However, the *pdf* and *cdf* can be calculated for any positive α and σ. For $[1<\alpha\leq 2]$ also the *Lomax distribution* is characterised by a finite mean value and infinite variance, and this α-region offers really heavy tailed characteristics.

In contrast to the Pareto type-1 distribution discussed before, here the *coefficient of variation* depends on both parameters. Lomax distributed numbers can be generated by *cdf* inversion via

$$X_{P_L} = \sigma\left(u^{-\frac{1}{\alpha}} - 1\right), \tag{2.51}$$

where $u = X_{U(0,1)}$ are uniformly distributed random numbers in $[0..1]$. To generate Lomax distributed numbers with a certain mean value we need to solve $X_{P_L}(\mu) = \mu\frac{X_{P_L}}{\mathrm{E}[X_{P_L}]}$ and get

$$X_{P_L}(\mu) = \mu(\alpha - 1)\left(u^{-\frac{1}{\alpha}} - 1\right), \tag{2.52}$$

where the dependence on the parameter σ is replaced by $\mu(\alpha-1)$, which achieves that σ is automatically chosen such that the intended mean value is achieved. In practice we use the generator for the generalised Pareto distribution presented next. The Lomax distribution is the special case that results for β=0 in equation 2.59. Figures 2.22 shows histograms of Lomax distributed example

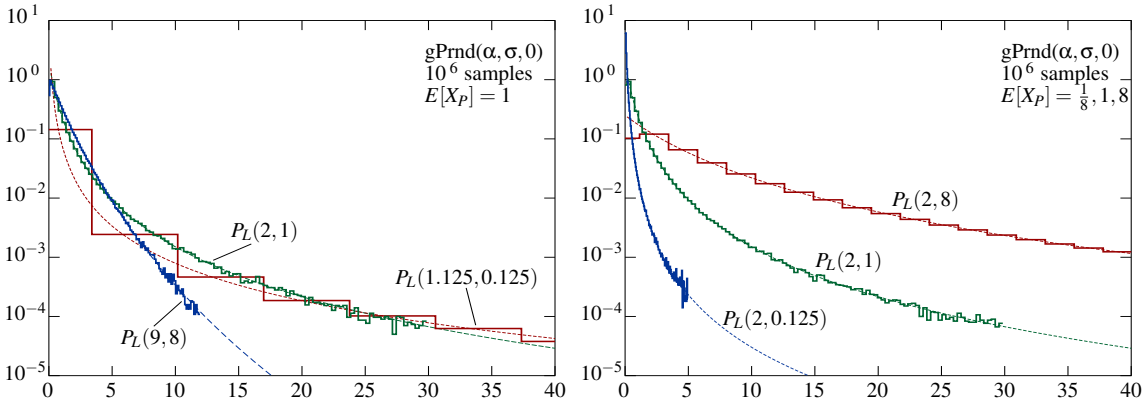


Figure 2.22: Histograms of different *Lomax* distributed samples for varying $\mathrm{E}[X_P]$ and α respectively.

sets (stepped) together with the according *pdf* (dashed). The close match proves that the generator actually generates samples with the intended distribution. That the $P_L(2,1)$ histograms shown in both figures differ in details results from different random sample sets (generation runs).

The histogram generation automatically chooses the bin size based on the sample variance ς^2, which evidently depends on the distribution's parameters, but also on the actual samples of the set. For infinite variance ($\alpha \leq 2$) the sample variance ς^2 becomes very changing, and thus, the bin sizes differ from set to set, even for huge sample-set sizes. The limited number of bins (here 100) cause in conjunction with the ς^2 dependent width of bins that the x-areas evaluated differ for different distributions. The x-axis has been chosen to show at least all bins of $P_L(2,1)$, the limit distribution for $Var(X) = \infty$ with $\mathrm{E}[X]$=1. Therefore, for $P_L(1.125,0.125)$ and $P_L(2,8)$ many bins (93 and 82 respectively) are outside the visible area. This expresses how heavy the tails actually become. Another divergence occurs for the first bin; its width is halve the size of the other bins, and therefore is its population not accurate.

We note that with the logarithmic scaled y-axis the negative exponential distribution's *pdf* would become a straight line. The concave curves visually prove that all Lomax distributions do have heavy tails. From Figure 2.22 we may assume that for $\alpha \to \infty$ the Lomax distribution approximates the negative exponential distribution $M(\mu)$, and for $\sigma \to 0$ the Lomax distribution approximates the Dirac delta function $\delta(0)$, as does the negative exponential distribution for $\mu \to 0$.

Generalised Pareto distribution

The *generalised Pareto distribution* (GPD) covers a wide range of power law distributions, and is also called *Pareto type-2*, why we indicate it as P_{II}. It results from shifting the Lomax distribution,

which starts at zero, up on the x-axis to re-introduce a lower bound β. Consequently, P_{II} is specified by three parameters; a *shape* parameter $\xi{=}\frac{1}{\alpha}$, a *scale* parameter $\varsigma{=}\frac{\sigma}{\alpha}$, and a *location* parameter $\mu'{=}\beta$, not to be mistaken with the distribution's mean value μ. The triple $[\xi,\varsigma,\mu']$ is used in the literature and with some statistical software tools that provide a GPD generator. For consistency with the preceding Pareto variants, we redefine the triple as $[\alpha,\sigma,\beta]$, and note the formal interchangeability. For comparability we show here the basics in both nomenclatures. Elsewhere we use the $[\alpha,\sigma,\beta]$-triple only, and presume that when using calculation tools all parameters are converted according to the nomenclature the used functions and procedures are defined in. If the nomenclature is not known, applying such a tool is inappropriate.

GPD – Pareto type-2 (P_{II})

pdf:
$$f_{P_{II}}(x) = \frac{1}{\varsigma}\left(1+\frac{\xi(x-\mu')}{\varsigma}\right)^{-(\frac{1}{\xi}+1)} = \alpha\sigma^{\alpha}\,(x+\sigma-\beta)^{-(\alpha+1)} \tag{2.53}$$

cdf:
$$F_{P_{II}}(x) = 1-\left(1+\frac{\xi(x-\mu')}{\varsigma}\right)^{-\frac{1}{\xi}} = 1-\sigma^{\alpha}\,(x+\sigma-\beta)^{-\alpha} \tag{2.54}$$

mean:
$$\mathrm{E}[X_{P_{II}}] = \begin{cases} \mu'+\frac{\varsigma}{1-\xi} = \beta+\frac{\sigma}{\alpha-1} & \alpha>1 \\ \textit{undefined} & \alpha\le 1 \end{cases} \tag{2.55}$$

variance:
$$Var(X_{P_{II}}) = \begin{cases} \frac{\varsigma^2}{(1-\xi)^2(1-2\xi)} = \frac{\alpha\sigma^2}{(\alpha-2)(\alpha-1)^2} & \alpha>2 \\ \infty & 1<\alpha\le 2 \\ \textit{undefined} & \alpha\le 1 \end{cases} \tag{2.56}$$

coefficient of variation:
$$c_X = \begin{cases} \frac{\sigma}{\beta(\alpha-1)+\sigma}\sqrt{\frac{\alpha}{\alpha-2}} & \alpha>2 \\ \infty & 1<\alpha\le 2 \\ \textit{undefined} & \alpha\le 1 \end{cases} \tag{2.57}$$

Again, for stability $\alpha{>}1$ is required, and for $[1{<}\alpha{\le}2]$ we get a finite mean value and infinite variance, which causes really heavy tailed, lets say 'chaotic', behaviour. The *coefficient of variation* c_X of P_{II} depends on all three parameters.

Again, P_{II} distributed numbers can be generated by *cdf* inversion, here via

$$X_{P_{II}} = \beta+\sigma\left(u^{-\frac{1}{\alpha}}-1\right), \tag{2.58}$$

where as usual $u = X_{U(0,1)}$ are uniformly distributed random numbers in $[0\,..\,1]$. To generate numbers with a certain mean value μ we need to solve $X_{P_{II}}(\mu) = \mu\frac{X_{P_{II}}}{\mathrm{E}[X_{P_{II}}]}$ and get

$$X_{P_{II}}(\mu) = \frac{\mu(\alpha-1)}{\beta(\alpha-1)+\sigma}\left(\beta+\sigma\left(u^{-\frac{1}{\alpha}}-1\right)\right). \tag{2.59}$$

Octave does not provide a precoded function to generate GPD samples, therefore we programmed the GPD generator outlined in algorithm 2.1 implementing equation 2.59. The GPD sample generator stated in algorithm 2.1 is called by

$$\texttt{gPrnd}(\alpha,\sigma,\beta,\mu),$$

which may be overdetermined if all parameters are given, and commentlessly ignores parameters given although not required. Parameters inserted as *zero* are automatically replaced by default values: the default $\alpha=2$ yields the chaos limiting case, the default $\sigma=\alpha-1$ yields unity shape ($\mu{=}\beta{+}1$), the default $\beta=0$ yields no lower boundary, and the default $\mu=\beta+\frac{\sigma}{\alpha-1}$ yields the mean defined by

Algorithm 2.1 *Octave* GPD generator code

```
function x=gPrnd(a,s,b,m)
do u=rand(1); until (u>0 & u<1); %exclude numeric zeros, assure symmetry
if !a a=2; endif %set default alpha=2 (the chaos limiting case)
if !b b=0; endif %set default beta=0 (no lower bound = Lomax distribution)
if !s s=abs(a-1); endif %set default sigma value that yields m-b=1
if !m m=b+s/(a-1); endif %calculate if not given (if m<=b mean is not definable)
if (a<=1) x=b+s*(u^(-1/a)-1); %use the generic form if no definable mean exists
elseif (b==0) x=m*(a-1)*(u^(-1/a)-1); %use Lomax with m adjusted s (ignores given s)
else x=m*(a-1)/(b*(a-1)+s)*(b+s*(u^(-1/a)-1)); %scale all parameters to meet given m
endif
endfunction
```

α, σ, β, which avoids any auto-scaling. The default settings allow to generate the chaos limiting case calling `gPrnd`(0,0,0,0), being a call without concrete parameters given. In order to be sure what is actually generated, the user is encouraged to always provide all intended parameters and to set only the unspecific to *zero*. The default setting lines in algorithm 2.1 may be out-commented to speed-up the performance, at the cost of potential numeric crashes in case of inconsistent parameter combinations.

In practice we might prefer to use precoded generators that are based on the even more general *Beta* or *Gamma* functions, for which highly optimized generation procedures are assumed. Using a precoded procedure we likely get unscaled samples and need to do the mean adjustment ourselves:

$$X_{P_{II}}(\mu) = \mu \underbrace{\frac{(\alpha-1)}{\beta(\alpha-1)+\sigma}}_{\text{const. factor}} \cdot \texttt{gprnd}\left(\tfrac{1}{\alpha}{=}\xi, \tfrac{\sigma}{\alpha}{=}\varsigma, \beta{=}\mu'\right),$$

where we assume that the provided generator function `gprnd` is defined in $[\xi, \varsigma, \mu']$.

To generate special cases of *Pareto distributions* we need to apply the specific parameter selections stated in table 2.1. For *Pareto type-1* we have to set $\sigma = \beta$, and for *Lomax* we need $\beta = 0$. Because the

Table 2.1: GPD generator calls for different Pareto distribution types

type	selection rule	$[\alpha, \sigma, \beta]$-based	$[\xi, \varsigma, \mu']$-based
Pareto type-1:	$\sigma = \beta$	$\texttt{gPrnd}(\alpha, \beta, \beta, \mu)$	$\texttt{gprnd}(\frac{1}{\alpha}, \mu\frac{\alpha-1}{\alpha^2}, \mu\frac{\alpha-1}{\alpha})$
Lomax:	$\beta = 0$	$\texttt{gPrnd}(\alpha, \sigma, 0, \mu)$	$\texttt{gprnd}(\frac{1}{\alpha}, \mu\frac{\alpha-1}{\alpha}, 0)$
Pareto type-2:	$---$	$\texttt{gPrnd}(\alpha, \sigma, \beta, \mu)$	$\texttt{gprnd}(\frac{1}{\alpha}, (\mu-\beta)\frac{\alpha-1}{\alpha}, \beta)$
$M(\mu)$:	$\alpha{=}\infty \,\&\, \beta{=}0$	$---$	$\texttt{gprnd}(0, \mu, 0)$

generalised Pareto distribution approximates the negative exponential distribution for $\beta = 0$ and $\alpha \to \infty$, we could use a $[\xi, \varsigma, \mu']$-based procedure to generate negative exponentially distributed $M(\mu)$, as shown by the last line in table 2.1. In *Octave* ∞ = `inf` is in principle supported, but not with the `gPrnd()` generator shown in algorithm 2.1. Inserting $\alpha =$ `inf` causes $x =$ NaN, meaning "not a number".

Figure 2.23 shows histograms and *pdf*'s of different Pareto distribution variants for equal shape α and mean $\mathrm{E}[X]$. The three parameters of the *generalised Pareto distribution* (P_{II}) allow us to preselect up to two parameters, and adjust the remaining only in order to achieve a certain mean value. If we fix the mean value, as done to produce figure 2.23, we may preselect one parameter

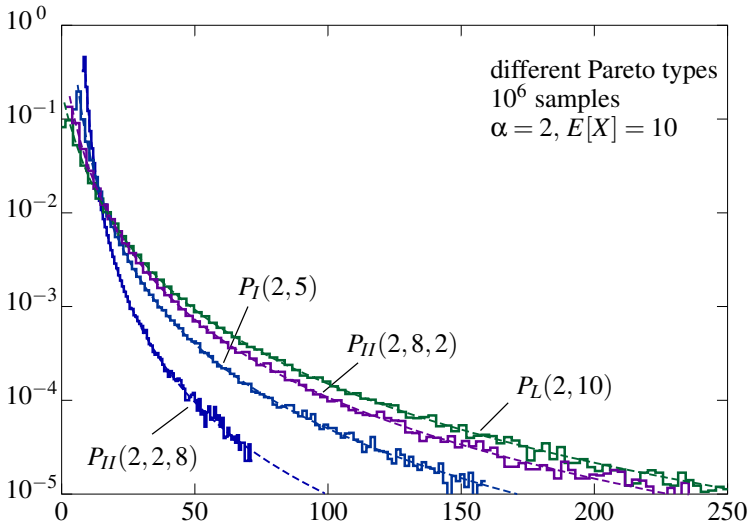


Figure 2.23: Histograms and *pdf* of different *Pareto* distributed sample sets comparing different Pareto types for same $\alpha = 2$ and $\mathrm{E}[X] = 10$

and still can use the other two to create different distributions that all have the same intended mean value. We chose the shape parameter $\alpha = 2$ because it is the chaos limiting case, and the mean value $\mathrm{E}[X] = 10$ to enable integer numbers for all example parameters. The influence of α on the curvature of the *pdf*'s decline has been shown by the left curves in figure 2.22, and the curves on the right show the influence of σ on the decline and consequently on the mean $\mathrm{E}[X]$ in case we keep α constant. From figure 2.23 we now can see the added *pdf* fitting freedom, but also that this comes at the price of changing lower bounds. Keeping the boundary β constant results in curves alike those shown in figure 2.22, only shifted along the x-axis by the value of β.

Considering that $P_I(2,5) = P_{II}(2,5,5) = \texttt{gPrnd}(2,5,5,10)$ and $P_L(2,10) = P_{II}(2,10,0) = \texttt{gPrnd}(2,10,0,10)$, we recognise in figure 2.23 how σ and β influence together the curvature of the *pdf*. We can derive, that for given shape α and mean $\mathrm{E}[X]$, the Lomax distribution is the heaviest, and that the tail weight decreases with decreasing σ and increasing β. Extrapolating this we may conclude that $P_{II}(\alpha, \sigma{\to}0, \beta{\to}\mu)$ becomes the *Dirac pdf* $\delta(\mu)$, representing the deterministic case. The opposite limit for $P_{II}(\alpha, \sigma{\to}\infty, \beta{=}0)$ is the non-existent infinite uniform distribution $U(0,\infty)$ with its degenerated *pdf* $f_{U(0,\infty)} = \frac{1}{\infty}$, equalling also the negative exponential distribution with infinite mean, $M(\infty)$.

More Pareto types and other power-law distributions

Concluding the Pareto distributions we note that also Pareto type-3 and type-4 exist,

$$F_{P_{IV}}(x) = 1 - \left(1 + \left(\frac{x-\beta}{\sigma}\right)^{\frac{1}{\gamma}}\right)^{-\alpha},$$

where Pareto type-3 results from type-4 for $\alpha{=}1$, and type-2 for $\gamma{=}1$.

We also note that various other distributions based on the power-law exist and that these might be used to model heavy tailed, potentially fractal, processes as well. A continuous power-law distribution is for example the *Student's t-distribution* and its special case, the *Cauchy distribution* with its peak at the median but undefined mean value. These distributions are rarely used with queueing systems, and thus not discussed in detail. The interested reader is referred to the respective literature on the topic. However, in the next subsection we present the *Beta distribution*. It is not useful for heavy tails, simply because it is double bounded, but its definition reveals that it is defined via two opposing power functions, which puts it in line here.

2.2.4 Beta distribution

If the probability is lower and upper bounded, we can use the *Beta distribution* $\mathcal{B}$ to model it. This distribution is quite versatile, in particular can it be used to model processes with a single maximum anywhere in between the boundaries as well as such with maxima at one or at both boundaries. Commonly it is defined for the interval $[0..1]$ by two shape parameters $\alpha, \beta > 0$.

Beta distribution ($\mathcal{B}$)

pdf:
$$f_{\mathcal{B}}(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 y^{\alpha-1}(1-y)^{\beta-1}\,dy} = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \qquad 0 \le x \le 1 \qquad (2.60)$$

cdf:
$$F_{\mathcal{B}}(x) = \frac{\int_0^x y^{\alpha-1}(1-y)^{\beta-1}\,dy}{\int_0^1 y^{\alpha-1}(1-y)^{\beta-1}\,dy} = \frac{B(x;\alpha,\beta)}{B(\alpha,\beta)} \qquad 0 \le x \le 1 \qquad (2.61)$$

mean:
$$\mathrm{E}[X_{\mathcal{B}}] = \frac{\alpha}{\alpha+\beta} \qquad (2.62)$$

moments:
$$\mathrm{E}\left[X_{\mathcal{B}}^{m+1}\right] = \prod_{k=0}^{m} \frac{\alpha+k}{\alpha+\beta+k} = \frac{\alpha+m}{\alpha+\beta+m}\mathrm{E}[X_{\mathcal{B}}^{m}] \qquad (2.63)$$

variance:
$$Var(X_{\mathcal{B}}) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \qquad (2.64)$$

coefficient of variation:
$$c_X = \frac{1}{\sqrt{\frac{\alpha}{\beta}(\alpha+\beta+1)}} \qquad (2.65)$$

Figure 2.24 shows different probability density functions of characteristic Beta distribution examples, and on the right the corresponding cumulative distribution functions. For $\alpha = \beta$ the

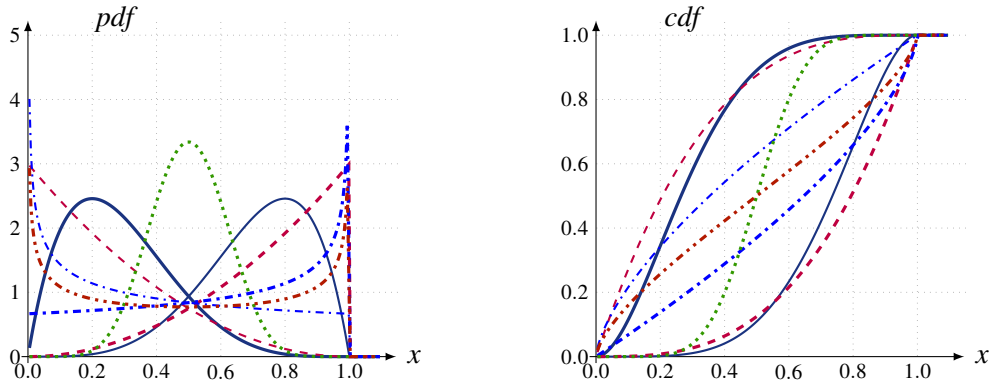


Figure 2.24: *pdf* and *cdf* of characteristic *Beta distribution* examples: $\mathcal{B}(2,5), \mathcal{B}(5,2)$ (solid), $\mathcal{B}(9,9)$ (dotted), $\mathcal{B}(3,1), \mathcal{B}(1,3)$ (dashed), $\mathcal{B}(1,\frac{2}{3}), \mathcal{B}(\frac{2}{3},1)$ (dot-dashed), $\mathcal{B}(\frac{2}{3},\frac{2}{3})$ (dot-dot-dashed).

distribution is symmetric with mean $\mathrm{E}\left[X_{\mathcal{B}(\alpha=\beta)}\right] = \frac{1}{2}$. If we swap $\alpha \leftrightarrow \beta$ the *pdf* becomes mirrored within the interval, identified in figure 2.24 by thick versus fine lines of same style. For $\alpha, \beta \le 1$ the distribution peaks at the according boundary, while for $\alpha, \beta > 1$ no peak at the according boundary occurs. A special case results for α=β=1 (not shown); in that case the Beta distribution equals the standard uniform distribution, $\mathcal{B}(1,1) \equiv U(0,1)$.

Characteristically different *pdf* shapes occur for $\mathcal{B}(>1,>1)$, $\mathcal{B}(\le 1,>1)$, $\mathcal{B}(>1,\le 1)$, and $\mathcal{B}(\le 1,\le 1)$. Particularly different to the common declining distribution functions are those with $\mathcal{B}(>1,\le 1)$, because for these the probability steadily increases toward the upper bound. Such a behaviour can hardly be approximated using unbound distributions. The *bathtub* shape (dot-dot-dashed) found for $\mathcal{B}(\le 1,\le 1)$ is a good model for product failures, either they fail quickly due to a

manufacturing fault, or last very long until ageing effects cause their failure. The shape we show for α=1, $\beta \leq 1$ (dot-dashed) can be used to approximate the packet-size distribution that results from splitting data-files into IP-packets, if $\frac{1}{\beta}$ equals the average number of IP-packets required per file.

The *pdf* and *cdf* in equation 2.60 and 2.61 use the *Beta function* $B(\alpha,\beta)$ to achieve $\int pdf(x)\,\mathrm{d}x = 1$ and $F_{\mathcal{B}}(\infty) = 1$. For the *cdf* also the *incomplete Beta function* $B(x;\alpha,\beta)$ is used. These functions are defined as

$$B(\alpha,\beta) = \int_0^1 y^{\alpha-1}(1-y)^{\beta-1}\,\mathrm{d}y, \quad \text{and} \quad B(x;\alpha,\beta) = \int_0^x y^{\alpha-1}(1-y)^{\beta-1}\,\mathrm{d}y.$$

Octave provides the function `beta(`α,β`)` to calculate the complete beta function $B(\alpha,\beta)$, and the function `betainc(`x,α,β`)` to calculate the *cdf* value $F_{\mathcal{B}}(x)$. The incomplete Beta function $B(x;\alpha,\beta)$ results if we multiply the two results, which we commonly do not required. Note again that names, here function names, may be irritating. For the `betainc(...)` function the integrated help on this function erroneously states that it calculates the *incomplete Beta function*. Analysing the function's calculation results we recognise that it actually calculates the *regularized beta function* $I_x(\alpha,\beta)$, which equals the *cdf* of the Beta distribution.

To generate such distributed numbers we use its relation to the *Gamma distribution* $\Gamma(k,\theta)$, because the generation of Gamma distributed samples is widely provided by software tools. It is known that $\frac{\Gamma(\alpha,\theta)}{\Gamma(\alpha,\theta)+\Gamma(\beta,\theta)} \sim \mathcal{B}(\alpha,\beta)$, independent of the scaling factor θ, because its influence is cancelled by the division. Consequently, if we generate independent samples for $X \sim \Gamma(\alpha,1)$ and $Y \sim \Gamma(\beta,1)$, we get Beta distributed samples by calculating

$$X_{\mathcal{B}(\alpha,\beta)} = \frac{X_{\Gamma(\alpha,1)}}{X_{\Gamma(\alpha,1)} + Y_{\Gamma(\beta,1)}}. \tag{2.66}$$

Alternative parametrisation

The *Beta distribution* may also be parametrised by its mean μ and a single shape parameter ν, which with some applications is called sample size:

$$\mu = \frac{\alpha}{\alpha+\beta}, \quad \nu = \alpha+\beta \quad \leftrightarrow \quad \alpha = \mu\nu, \quad \beta = (1-\mu)\nu.$$

Such defined we get quite simple equations,

$$\mathrm{E}[X_{\mathcal{B}}] = \mu, \qquad Var(X_{\mathcal{B}}) = \frac{\mu(1-\mu)}{\nu+1} \qquad c_X = \frac{1}{\sqrt{\frac{\mu}{1-\mu}(\nu+1)}}. \tag{2.67}$$

This is useful if we model arrival intervals or service times bound to a normalised interval $[0\,..\,1]$. Using this parametrisation, the original α and β parameters are contrarily adjusted, such that different normalized mean values $0 \leq \mu \leq 1$ are achieved, without changing $\nu = \alpha+\beta$.

Extension to general interval $[a\,..\,b]$

In practice, where the lower and upper bounds a, b likely are not *zero* and *one*, we need to shift and stretch the Beta distribution by substituting

$$x^{\mathcal{B}} = \frac{x-a}{b-a}.$$

Inserting this substitution in the equations above we get the four parameter version $\mathcal{B}_4$, with

$$pdf: \qquad f_{\mathcal{B}_4}(x) = f_{\mathcal{B}}\left(\frac{x-a}{b-a}\right) = \frac{(x-a)^{\alpha-1}(b-x)^{\beta-1}}{(b-a)^{\alpha+\beta-1}B(\alpha,\beta)}, \tag{2.68}$$

$$mean: \qquad \mathrm{E}[X_{\mathcal{B}_4}] = a + (b-a)\,\mathrm{E}[X_{\mathcal{B}}] = \frac{\alpha b + \beta a}{\alpha+\beta}, \tag{2.69}$$

$$variance: \qquad Var(X_{\mathcal{B}_4}) = (b-a)^2\, Var(X_{\mathcal{B}}) = \frac{(b-a)^2\,\alpha\beta}{(\alpha+\beta)^2\,(\alpha+\beta+1)}. \tag{2.70}$$

Evidently can this shifting and stretching be equally applied on generated samples. Therefore, it is sufficient to generate samples for $\mathcal{B}(\alpha,\beta)$ and subsequently calculate

$$X_{\mathcal{B}_4(a,b,\alpha,\beta)} = a + (b-a)\,X_{\mathcal{B}(\alpha,\beta)}, \tag{2.71}$$

to get Beta distributed samples for any interval $[a\,..\,b]$.

2.2.5 General distribution

Evidently, a *general distribution* is nowhere defined. However, to express that properties are valid for any distribution, the place-holder abbreviations *G* and *GI* have become common. Still, to get numeric results it may be necessary to insert an actual distribution function.

General independent (*GI*)

The *general independent* abbreviation *GI* expresses that the events have to be independent and identically distributed. This constraint prohibits correlation of events as well as any time dependent distribution. Events are *independent* if no correlation among events exists. Concerning the random process this demands

$$\mathrm{P}[A(i) = x \cap A(j) = y] \stackrel{!}{=} \mathrm{P}[A(i) = x] \cdot \mathrm{P}[A(j) = y]\,. \tag{2.72}$$

That the events occurrences need to be *identical* forces that the distribution must not change over time. This *time-invariance* implies that the process is independent of system changes, meaning in consequence that the modelled arrival or service process must not depend on the current system state. For priority based scheduling systems and congestion controlled flows this condition is never ever fulfilled. Also, the *Engset model* does not fulfil this condition because of its state dependent arrival rates caused by the finite population.

Luckily, for the service process the restriction forcing independent and identically distributed events is mostly not required in the course of analytic evaluations. Consequently, we can use the more general results found for $./G/\ldots$, which evidently cover the corresponding $./GI/\ldots$ scenario. An extreme example for state dependent service times is *processor sharing*, discussed later on for example with multi-queue systems (section 4.1) and processor sharing in general (section 3.3.3).

General (*G*)

The *general* abbreviation *G* expresses that absolutely any distribution function may be inserted. Also distribution functions that include dependences on the past and the current system state are allowed. Absolutely no restrictions apply, and thus, no more can be said about this place-holder.

The place-holders *G* and *GI* are widely used to express the generality of distribution independent properties of systems, which of course are of prime interest if the actual distribution is not known, as it is often the case with random systems in practice. To assure their universality such properties need to be mathematically derived without inserting any distribution function.

2.3 Fitting a distribution to match a sample sequence

In the previous two sections we introduced some distributions. Commonly, traffic occurring in practice is not generated by a determined stochastic processes that implements a specific distribution function; such only applies for modelling and testing purposes. Real traffic characteristics are determined by user actions and the design of the technical processes, being the designed response of systems to different events. Concerning data transmission it is common practice to assume Poisson distributed user actions. The responses of the technical components invoked to perform data transmission is usually state and application dependent. In general many processes work together in order to perform the action demanded by a user, and a cascade of components and interfering demands influence the resultant data traffic's characteristic.

The huge number and heterogeneity of systems and processes involved, makes it intractable to precisely state the properties of *traffic flows* based on the component's individual responses. Still, different applications cause typical traffic characteristics. These are not precisely determined, they comprise a certain level of uncertainty. Therefore, a suitable approach to identify these characteristics is to record representative sample sequences (*traces*) and to find (match) distribution functions, which resemble the characteristics not precisely but in a maximally general sense.

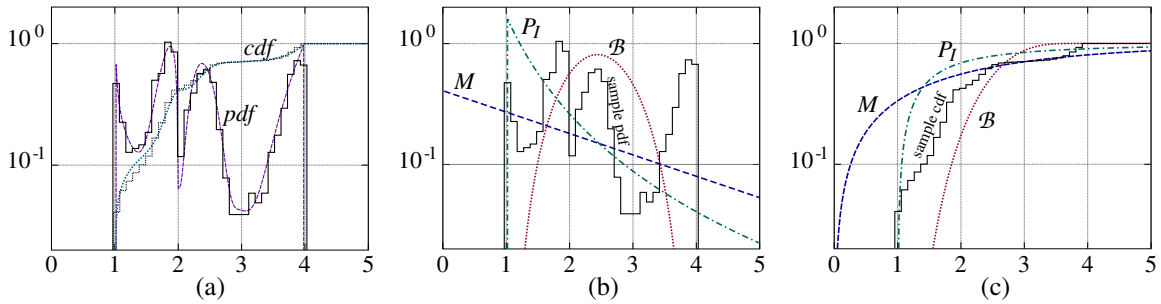


Figure 2.25: Mapping a distribution to composed example: (a) example trace's histogram (solid), *pdf* (dashed), *cdf* (dotted), and the distribution's *pdf* (b) and *cdf* (c) matching for negative exponential $M(\mu)$ (dashed), power law $P_I\left(\frac{\mu}{\mu-1}, a\right)$ (dash-dotted), and $\mathcal{B}\left(\frac{(b-\mu)(b-a)}{\varsigma^2}, \frac{(\mu-a)(b-a)}{\varsigma^2}, a, b\right)$ (dotted).

In principle any distribution function can be mapped to any trace, as shown by the example depicted in figure 2.25. The example distribution in 2.25.a is a composite of four Beta distributions, $\mathcal{B}_4(0.4,1,1,4) + \mathcal{B}_4(8,2,1,2) + \mathcal{B}_4(4,6,2,3) + \mathcal{B}_4(4,1,3,4)$, yielding four peaks. To match the distribution functions inserted in 2.25.b and .c we use the sample mean $\mathrm{E}[X_n] = \mu = 2.45395$, the bounds $a = 1$, $b = 4$, and the sample's standard deviation $\varsigma = 0.92284$; in this order, as far as parameters can be chosen to fit the particular distribution. Evidently, a perfect match cannot be achieved using a single distribution with one peak only. Also, when matched by differently bounded distributions an inevitable divergence occurs. Commonly, if several peaks exist we should decompose the example into a composition of matched distributions (one per peak) in order to achieve a good fit. This approach is out of the scope here, where we intend to find a representative distribution, which fits such that the analysis yields feasible results, no more.

The number of relevant parameters is commonly rather small, in particular if we only need to identify the typical behaviour. Thus, we need to ask if using a closely matching, potentially complex, distribution function is worth the effort. Particularly, if we cannot say for sure that the available traces are representative. Commonly, few estimated parameters from limited observations are sufficient to identify the mean system properties with a reasonable *effort to accuracy balance*. These typical properties can than be used to select a stochastic process that appropriately models the in its details hidden system causing the observed trace, with marginal loss of accuracy if the observed trace is a possible outcome of the chosen process.

Experience with typical *pdf* shapes is in this context very advantageous to visually identify the distribution type from *histograms*, independent of the actual numeric values. If we manage to select the correct distribution type for the matching, it is likely that even though we achieve no perfect matching also the atypical properties can be studied with sufficient significance.

Considering that (i) a finite sample set, which any recorded trace evidently is, cannot perfectly characterise the process it results from, and (ii) that complex distribution functions raise the effort dramatically, we focus on methods to identify the most simple distribution function that sufficiently approximates the available estimates on distribution characteristics. For analytic studies we commonly assume that ingress traffic and service time characteristics match the negative exponential distribution. The other widely used simplification, the *independence assumption*, we try to avoid by modelling chained network elements as one entity. Via simulation studies we can than analyse how sensitive the studied systems (*network entities*) respond to diverging distribution characteristics.

2.3.1 Comparing of distributions

To assess the above mentioned matching errors we need some method to compare distributions independent of their origin. Here in particular, to identify how sever a models distribution function diverges from the distribution of the samples (traces) that we match the model to.

Moments comparing

Any distribution is precisely defined by all its moments. This fact can be used to compare distributions. The more moments match, starting with the first and ascending to higher moments without omissions, the better is the analogy. However, all may not be considerable, for example due to the potentially infinite number of moments that finite traces can define.

Of particular interest is the highest moment-order considered for matching, the so called *matching order*. Typically, second order matching is sufficient to achieve adequately useful models for most analyses. Methods to achieve this are presented in section 2.3.4. Three moments matching is likely excessive, while a simple first order matching is rather sufficient for mean value analysis of independent systems only.

Table 2.2: First two moments of the examples shown in Figure 2.25

	trace	M	P_I	$\mathcal{B}$
mean:	2.45395	2.45395	2.45395	2.45395
variance:	0.85163	6.02187	∞	0.85163

Looking at the second moment divergences shown in table 2.2 for the examples depicted in figure 2.25, we recognise that the matched negative exponential distribution shows a too high variance. The matched Pareto distribution considers the lower bound; however, the infinite variance is excessive and will heavily hamper any analysis based on this model. The best match results for the Beta distribution, which is evident, because here we achieve a match not only in both moments but also in both boundaries. A potential distribution for a better match by an unbounded distribution should be the Erlang distribution. A method to perform this is discussed later on in section 2.3.4.

How severe a divergence in moments actually is, is difficult to assess because that depends on the circumstances, meaning the intended analysis focus. An interesting method is for example the *effective bandwidth* approach: the process is approximated by a negative exponential distribution with minimal mean square deviation of all moments, i.e., $\min\left(\sum(\mu^{[k]} - \hat{\mu}^{[k]})^2\right)$. In that case lower and higher moments diverge individually, and commonly no moment will be matched exactly, including the means. The utility of such an approximation needs to be validated case by case.

Graphical comparing

The *pdf* or *cdf* precisely define a distribution. Therefore, comparing these functions with those defined by the trace yields an option to graphically assess the analogy. The *empirical* distribution functions $f_n(x)$ and $F_n(x)$ we get from the trace X_n via its histogram

$$\nu_i(X_n) = \sum_{j=1}^{n} \left(1 \,\middle|\, (x_i - \tfrac{\Delta_x}{2}) < x_j \leq (x_i + \tfrac{\Delta_x}{2})\right) \qquad i = 1\,..\,m \tag{2.73}$$

as

$$f_n(x_i) = \frac{\Delta_x}{x_{max} - x_{min}}\, \nu_i(X_n) \tag{2.74}$$

$$F_n(x_i) = \sum_{j=0}^{i} \Delta_x\, f_n(x_j) \tag{2.75}$$

where Δ_x is the bin-width, x_i are the bin-centres, and $m = \frac{x_{max} - x_{min}}{\Delta_x}$ is the number of bins. The histogram $\nu_i(X_n)$ states the number of samples from X_n that fall into the i^{th} bin. The scaling factor $\frac{\Delta_x}{x_{max} - x_{min}}$ normalises the histogram $\nu_i(X_n)$ such that it becomes the *empirical pdf* with $\sum_{i=1}^{m} f_n(x_i) = 1$, where $f_n(x_i)$ are the bin probabilities. The *empirical cdf* we get from $F_n(x_i) = \int_0^x f_n(x_i)\,\mathrm{d}x$, which reduces to the accumulation of bins, being their width weighted by their probability.

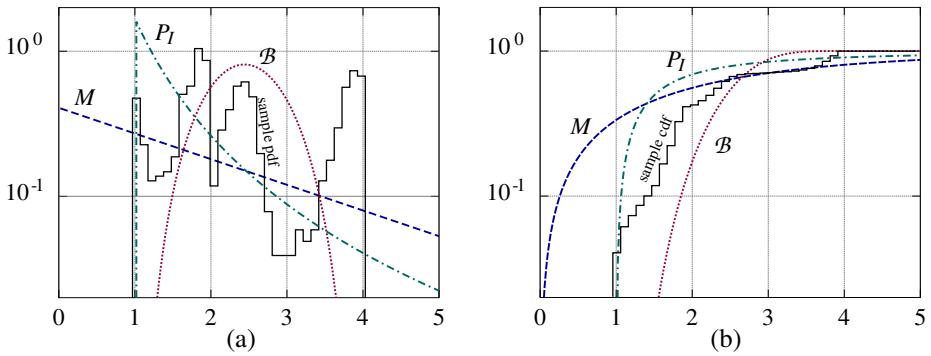


Figure 2.26: Empirical and matched distribution functions: (a) *pdf*s, (b) *cdf*s

Figure 2.26 repeats figure 2.25.b and 2.25.c in reasonable scale to discuss what actually can be visually seen and how divergences may be calculated. First, we recognise that no mapping fits well. This has been intended to highlight that precise mapping is often not essential for a useful model. The primary problem cause the multiple peaks, the secondary the boundedness. Particularly, the *pdf* lobe at the upper boundary becomes badly neglected. Divergences are commonly more visible comparing *pdf*s, while the severity of divergences can be better assessed from comparing *cdf*s. Therefore, we prefer to use *pdf* comparing for visual similarity assessing, and *cdf* comparing for the numeric *goodness of fit* analysis presented shortly.

A simple *cdf* comparing that can be performed manually is *quantiles comparing*. Actually, we compare the quantiles of the complementary *cdf*, $\overline{F}(x) = 1 - F(X)$ (x-axis) with the quantiles calculated from the available sample (y-axis), as shown in figure 2.27.a. A point at (x,y) states the quantile of the sample plotted against the same quantile of the distribution function. This results in a parametric curve with the parameter being the index of the quantiles interval compared point by point. If the two distributions are similar, the points in the *quantiles-quantiles* plot will approximately lie on the line $y{=}x$. If the distributions are linearly related, the points will approximately lie on a diverging line. Thereby the shapes of distributions can be graphically compared, allowing a graphical assessment on whether the distribution's tails are comparably heavy for increasing quantiles.

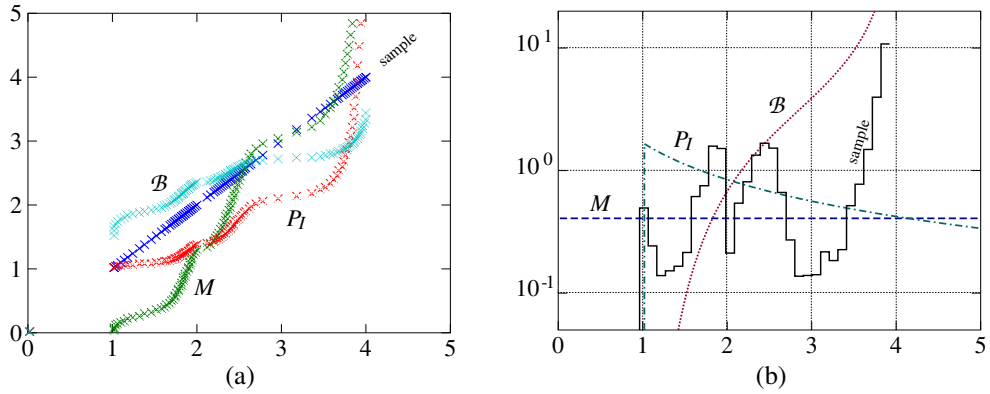


Figure 2.27: Derived graphs (a) quantiles-quantiles plot, (b) the failure function $h(x)=\frac{f(x)}{1-F(x)}$

Another property that can be compared graphically is the function $h(x) = \frac{f(x)}{1-F(x)}$ shown in figure 2.27.b, which represents a conditioned probability and is well known in reliability theory and survival analysis as *failure function*. Simply speaking, an increasing $h(x)$ states that the failure probability rises over the time waiting for a failure to occur, whereas a decreasing $h(x)$ indicates a system that likely fails early but rarely if for long time no failure occurred. Obviously, a good match demands similar tendency. However, the example chosen shows that for multi-modal distributions the tendency is no unique. However, the tendency is positive, which is actually a necessity for any bounded process. If the time till the next failure cannot approach infinity, the probability that it occurs must rise with the time awaiting it.

Numeric match validation

The *goodness of fit* criterion is used for judging the matching quality, here between the fitted cumulative distribution function $F_X(x)$ and the empirical distribution function $F_n(x)$ derived from the available sample. The underlying assumption is the null hypothesis, meaning the hypothesis that the existing sample can be an outcome of the matched distribution [44]. Tests for the equality of continuous one-dimensional probability distributions provide a numeric *goodness of fit* criterion based on comparing empirical and analytic distribution functions.

The ***Kolmogorov-Smirnov test*** [45] quantifies the maximum distance between $F_n(x)$ and $F_X(x)$. It is a general, non-parametric method for comparing, applicable for any continuous distribution, sensitive to differences in both, location and shape of the *cdf*s compared.

$$d = \sup_x |F_n(x) - F_X(x)| \tag{2.76}$$

The metric d is the maximum vertical distance between the empirical *cdf* $F_n(x)$ derived from the sample and the fitted *cdf* $F_X(x)$, calculated as $\sup_x \{\dots\}$ = the *supremum* over all x. However, it does not tell how good a fitting in average is.

The ***Cramér-von Mises criterion*** is named after Harald Cramér and Richard Edler von Mises who first proposed it in 1928/1930, reviewed in [46], and evaluates the area between $F_n(x)$ and $F_X(x)$.

$$\omega^2 = \int_{-\infty}^{\infty} [F_n(x) - F_X(x)]^2 \, \mathrm{d}F_X(x) \tag{2.77}$$

The bigger ω^2 is, the less accurately fits $F_X(x)$ the sample. Based on tables the hypothesis that the sample may have resulted from a distribution equal the fitted one is accepted or rejected. Evidently is $\omega^2 \geq 0$, and because of the square any potential compensation of positive and negative divergences is neglected. Thus, it is quite conservative in validating the matching quality.

Pearson's chi-squared test [47] is similar, but based on quantized sample frequencies, and seems to be the best suited metric for a simple matching quality validation.

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \tag{2.78}$$

It directly uses the histogram, where O_i is the number of samples that fall into bin i, $i \in [1\,..\,k]$, which is comparable to the *pdf* scaled by $\frac{n}{k}$. The number of samples E_i that the bin should contain if the fitted distribution X equals the empirical distribution X_n is best calculated by $E_i = n\,[F_X(x_i) - F_X(x_{i-1})]$ from the fitted *cdf*, where the x_i are the bin boundaries ($x_0 = 0$, $x_{k+1} = \infty$). The test may fail if the expected frequency per bin is too small ($<$10). This can be avoided by adjusting the bin sizes such that $E_i > 10\ \forall_i$. If this is not possible, Yates's correction for continuity

$$\chi^2_{\text{Yates}} = \sum_{i=1}^{k} \frac{(|O_i - E_i| - 0.5)^2}{E_i}$$

may be used to improve the quality of the test criterion. This correction prevents an overestimation of statistical significance for small data. Unfortunately, Yates' correction tends to over-correct the criterion and thus should be used with particular care.

The ***Anderson-Darling test*** [48], in its basic form, is distribution independent and can be generally applied. To calculate the criterion A^2 (equation 2.79) we first have to sort the outcome y_i of the sample in ascending order, such that $y_i \leq y_{i+1}\ \forall_i$.

$$A^2 = -n - \sum_{i=1}^{n} \frac{2i-1}{n} \left[\ln(F_X(y_i)) - \ln(1 - F_X(y_{n-i+1}))\right] \tag{2.79}$$

Proving that this criterion establishes a goodness of fit validation is left to the experts, as are the modifications to the test statistic and the critical values required to test if a sample may be fitted by a normal distribution, a negative exponential distribution, or some other distribution family. See for example Pearson and Hartley (1972) [49] and Scholz and Stephens (1987) [50]. When applied to test if a set of data (sample) is normal distributed, the *Anderson-Darling* test is said to be one of the most powerful statistical tools to detect outliers (departures from normality).

These methods provide a scalar value that reflects in some way to which extent the fitted distribution diverges. Comparing the achieved *goodness of fit* criterion with the *critical values* derived for different significance levels we may assess whether the available sample may have resulted from the fitted distribution or not. No test is capable to suggest potential improvement, and thus, in their basic form they cannot replace a visual validation. The generalisation mentioned above upon testing if a distribution family can be fitted to a sample by adjusting some parameters, is possible. However, the required adoption of the criterion or the critical values, are out of the scope. Please refer to the rich literature on test modifications that achieve this generalisation for particular distribution families, for example in [51–53].

2.3.2 Fitting a negative exponential distribution

The negative exponential distribution defines mathematically the boundary between *smooth* and *bursty* one sided distributions. Therefore, it is the obvious candidate for matching, in particular if we have no information on the distribution's shape. The negative exponential distribution is fully defined by its mean value, and thus, to perform the matching, we only need to set the first moment μ equal the sample mean $\mathrm{E}[X_n]$.

$$\mu = \mathrm{E}[X_n] = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.80}$$

$$\tilde{X} = M(\mu) = \mu \cdot \ln(U_{[0,1]}) \quad \Rightarrow \quad f_{\tilde{X}(\mu)} = \frac{1}{\mu}\, e^{\frac{x}{\mu}}, \quad F_{\tilde{X}(\mu)} = 1 - e^{\frac{x}{\mu}}$$

The most important advantage of negative exponential distribution mapping is the mathematical practicability. No equivalently applicable and handsome modelling approach exists for the analytic evaluation of systems. It reduces system analysis to the analysis of Markov chains. However, the difference between modelled and realistic system properties can become huge if the properties heavily depend on the variance. Therefore, different methods have been proposed to adjust the negative exponential distribution modelling in a way that enables better prediction of variance dependent properties. One of them being the *effective bandwidth* approach extensively studied with the *asynchronous transfer mode* (ATM) technology.

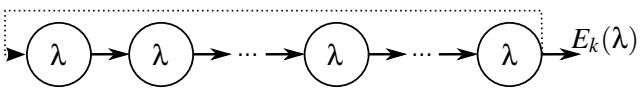
2.3.3 Moments matching

To achieve a model that fits an empirical distribution more precisely we need to fit more moments. The simplest method to achieve a two moments matching degree, is to directly find the parameters that satisfy the moments equations presented in sections 2.1 and 2.2, repeated here for convenience.

$$\begin{aligned}
\mathrm{E}[X_{E_k}] &= \frac{k}{\lambda} & Var(X_{E_k}) &= \frac{k}{\lambda^2} \\
\mathrm{E}[X_{H_k}] &= \sum_{j=1}^{k} \frac{\alpha_j}{\lambda_j} & Var(X_{H_k}) &= 2\sum_{j=1}^{k} \frac{\alpha_j}{\lambda_j^2} - \left(\sum_{j=1}^{k} \frac{\alpha_j}{\lambda_j}\right)^2 \\
\mathrm{E}[X_U] &= \frac{a+b}{2} & Var(X_U) &= \frac{(b-a)^2}{12} \\
\mathrm{E}[X_P] &= \frac{\alpha\beta}{\alpha-1} & Var(X_P) &= \frac{\alpha\beta^2}{(\alpha-2)(\alpha-1)^2} \\
\mathrm{E}[X_{P_L}] &= \frac{\sigma}{\alpha-1} & Var(X_{P_L}) &= \frac{\alpha^2\sigma}{(\alpha-2)(\alpha-1)^2} \\
\mathrm{E}[X_{P_H}] &= \beta + \frac{\sigma}{\alpha-1} & Var(X_{P_H}) &= \frac{\alpha\sigma^2}{(\alpha-2)(\alpha-1)^2}
\end{aligned}$$

Most of them have their moments defined by two parameters, and therefore the matching of two moments is precisely defined. Actually, if a distribution is defined by two parameters only, precise matching is restricted to two moments, because else the equation system to solve is overdetermined. Still, an approximate higher order matching can be achieved, for example by minimising the mean square moments divergences. However, in that case no moment is matched precisely.

Let us for example fit the *Erlang distribution* E_k (figure 2.28) to the example we already know, depicted in figure 2.25.a.



$$f_{E_k}(x) = \lambda \frac{(\lambda x)^{k-1}}{(k-1)!} e^{-\lambda x}$$
$$F_{E_k}(x) = 1 - \sum_{i=1}^{k} \frac{(\lambda x)^{i-1}}{(i-1)!} e^{-\lambda x}$$

Figure 2.28: The Erlang-k model

$$\begin{aligned}
\mu_n = \frac{k}{\lambda} \quad &\Rightarrow \quad k = \mu_n\lambda \\
\varsigma_n^2 = \frac{k}{\lambda^2} = \frac{\mu_n}{\lambda} \quad &\Rightarrow \quad \lambda = \frac{\mu_n}{\varsigma_n^2}, \quad k = \frac{\mu_n^2}{\varsigma_n^2}
\end{aligned} \tag{2.81}$$

Solving the equations for the example's empirical first two moments, μ_n=2.45395, ς_n^2=0.85163, yields λ=2.88147, k=7.07099. By intuition we first select the closest integer for k, here $\hat{k} = 7$, and adjust $\hat{\lambda} = \frac{\hat{k}}{\mu_n} = 2.85254$ to compensate the mismatch. The result is an imperfect fit; while $\mu_n = \frac{\hat{k}}{\hat{\lambda}} = 2.45395$ perfectly fits, $\varsigma_n^2(\hat{k}{=}7) = 0.86027$ slightly diverges from the prospected value. If

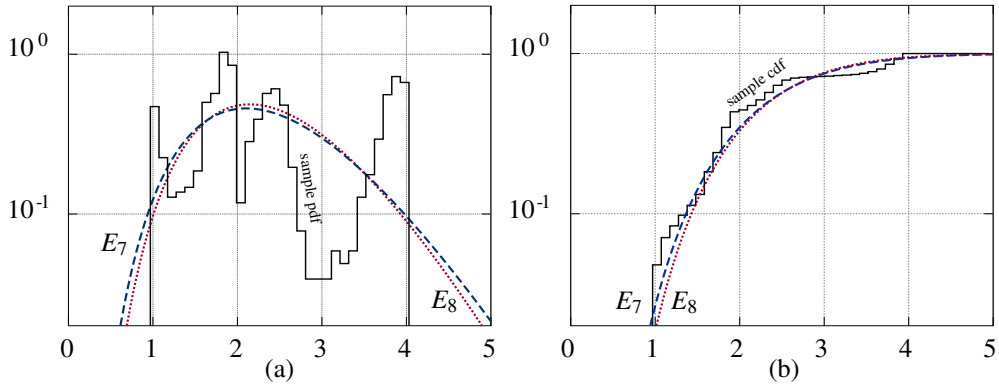


Figure 2.29: Fitted E_k distribution functions: (a) *pdf*s, (b) *cdf*s; k=7 (dashed), k=8 (dotted)

we choose $\hat{k} = 8$ and adjusted $\hat{\lambda} = 3.26005$ we get a worse second order fit $\varsigma_n^2(\hat{k}{=}8) = 0.75273$, which indicates that $|k - \hat{k}|$ should be minimised to achieve the best possible E_k fitting. Figure 2.29 depicts how good the two E_k models fit the empirical example distribution. Particularly the *cdf*-match looks better than for any model considered before (figure 2.25 and 2.26). Evidently, the different peaks and the boundaries of the example are not modelled. Still, also the *pdf*-fitting appears to be in average a closer match, where the parts outside the boundaries compensate the missed peaks of the example at its boundaries.

The moments mapping method is applicable for any distribution, as long as an according number of moments is defined. However, if the available parameters are restricted, for example to being integer, we cannot always achieve a perfect mapping.Anyhow, before we can fit a distribution to a sample, we need to decide upon the distribution that we want to use for the model. A look on the empirical *pdf* and other useful graphs (see section 2.3.1 for details) unveils the typical properties of the commonly hidden process, far more revealing than few moments alone. These properties are useful for a clever identification of the distribution family that yields a model with comparable behaviour. Beyond the thereby achievable degree of fitting goodness, we need to retard to distributions that provide a potentially unlimited number of parameters.

2.3.4 Fitting the H_k and Cox_k model to extreme distributions

Out of the examples listed in the previous section 2.3.3, only the *hyper-exponential distribution* H_k provides a selectable degree of fitting freedom. With every additional phase we get two additional parameters, a new λ_i and a new splitting factor α_i, as shown in figure 2.30. Using a H_k with as many

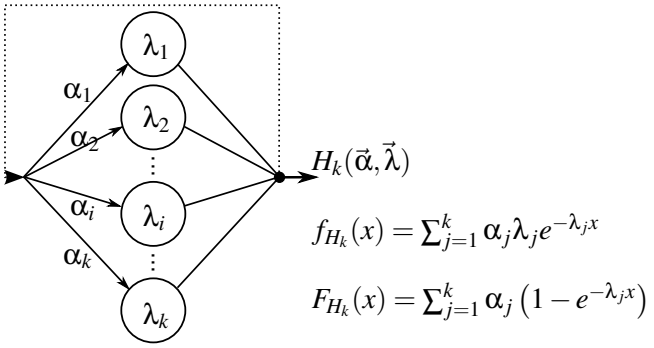


Figure 2.30: The hyper-exponential model

phases as there are moments we want to match, yields an under-determined system of equations. This allows perfect fitting for integer k, but offers an infinite number of equally perfect models.

Commonly the *balancing* condition $\frac{\alpha_i}{\lambda_i}=const$ is added to solve the under-determination problem. However, to minimise the complexity we desire the smallest possible model, meaning the smallest k that provides a sufficient fit. Because $\sum\alpha_i=1$ must be fulfilled, only $k-1$ out of all α_i may be adjusted. The number of adjustable parameters for any H_k is thus $2k-1$. Therefore, H_2 is sufficient for three moments matching because H_2 readily provides three parameters: two phase-rates λ_1, λ_2, and the splitting ratios α, $1-\alpha$. In general, a H_k model with $k=\lceil\frac{n+1}{2}\rceil$ phases is needed to match the first n moments, where for even n one parameter is freely chosen.

H_k fitting scheme

Let us consider now the case where the *hyper-exponential* H_k model is used to approximate a heavy-tailed distribution, for example a *Lomax distribution* $P_L(\alpha,\sigma)$, where only for $k\to\infty$ a perfect match is in theory possible. The moments matching method cannot be applied because for small α the higher moments of the *Lomax distribution* are infinite or not defined at all.

The approach presented in [54], which we here outline, is applicable for any distribution with *monotonous* decreasing *pdf*. The complementary *cdf* $F^c=1-F_{P_L}$ is step by step adjusted by subsequent adding of phases that reduces the remaining difference. We start with the weighted negative exponential distribution that fits the tail as accurate as intended by selecting a sufficiently large matching point c_1 because the tail of the negative exponential distribution is heavier for bigger λ. Step-by-step we than chose geometrically *declining* matching points $c_i=\frac{c_{i-1}}{d}$, where $d>1$ is required and $d=10$ is used in [54], such that the subsequently added phases adjust step-by-step lower parts of the power law distribution (right to left). The process terminates at an a priori chosen $i=k$, which causes that the goodness of fit is not implicitly granted (has to be checked independently).

The first phase of the H_k, defined by λ_1 and α_1, we get from solving

$$\begin{aligned}&\alpha_1\,e^{-\lambda_1 c_1}=F^c(c_1)\qquad\text{and}\qquad\alpha_1\,e^{-\lambda_1 bc_1}=F^c(bc_1)\\&\Rightarrow\quad\lambda_1=\frac{\ln(F^c(c_1))-\ln(F^c(bc_1))}{c_1(b-1)},\quad\alpha_1=F^c(c_1)\,e^{\lambda_1 c_1},\end{aligned}\tag{2.82}$$

where $1<b<d=\frac{c_{i-1}}{c_i}$ needs to be fulfilled. Having an intermediate model *ccdf*, we calculate the remaining *ccdf* divergence $F_i^c(c_i)$ at the next matching points

$$F_i^c(c_i)=F_{i-1}^c(c_i)-\sum_{j=1}^{i-1}\alpha_j\,e^{-\lambda_j c_i}\qquad\text{and}\qquad F_i^c(bc_i)=F_{i-1}^c(bc_i)-\sum_{j=1}^{i-1}\alpha_j\,e^{-\lambda_j bc_i},\tag{2.83}$$

where $F_1^c=F^c$, to get the next parameters pair, λ_i,α_i, from

$$\begin{aligned}&\alpha_i\,e^{-\lambda_i c_i}=F_i^c(c_i)\qquad\text{and}\qquad\alpha_i\,e^{-\lambda_i bc_i}=F_i^c(bc_i)\\&\Rightarrow\quad\lambda_i=\frac{\ln(F_i^c(c_i))-\ln(F_i^c(bc_i))}{c_i(b-1)},\quad\alpha_i=F_i^c(c_i)e^{\lambda_i c_i}.\end{aligned}\tag{2.84}$$

This procedure is repeated until $i=k-1$. The last phase k we get from

$$\alpha_k=1-\sum_{j=1}^{k-1}\alpha_j\quad\Rightarrow\quad\lambda_k=\frac{\ln(\alpha_k)-\ln(F_k^c(c_k))}{c_k},\tag{2.85}$$

because $\sum\alpha_i=1$ needs to be achieved. If all weights are positive, $\alpha_i>0$, and the λ_i are well separated, than we should have a good fit. Note that across the matched area the *cdf* of the H_k needs to be above the *cdf* of P_L, at least at all matching points, due to the heaviness of the power law tail. The remaining error is

$$\Delta(x)=|F_{H_k}^c(x)-F_{P_L}^c(x)|=|F_{H_k}(x)-F_{P_L}(x)|,\tag{2.86}$$

which approaches zero for $c_1, k \to \infty$ due to the *log-convexity* of both *ccdf*s. Actually, a H_∞ can precisely model any heavy-tailed distribution with monotonous decreasing *pdf* because latter condition causes the therefore required log-convexity. Any such distribution can be split into Markovian phases. However, the infinite number of phases limits the applicability to finite approximations. Any goodness of fit can in theory be achieved, comparable to the splitting of functions into a potentially infinite number of spectral components.

Figure 2.31 depicts the above mentioned example. We fit a hyper-exponential distribution with four phases ($k = 4$) to the empirical *cdf* calculated from a rather small sample ($n = 2500$) drawn from the *Lomax* distribution with $\alpha = 2$ and $\sigma = 1$, for which $\mathrm{E}\left[P_{L(2,1)}(x)\right] = 1$. The first matching

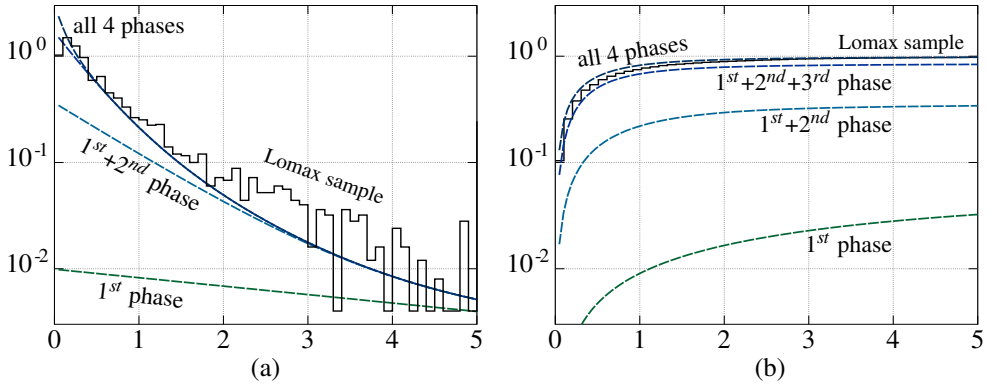


Figure 2.31: H_4 distribution step-by-step fitted to fractal $P_L(2,1)$: (a) *pdf*s, (b) *cdf*s

point has been set to $c_1 = 6$ (outside the depicted area), the declining factor to $d = 8$, and the fitting factor to $b = 2$ (as recommended in [54]). Some experimenting with initial parameters was necessary to determine a useful selection for d. Too small as well as too big d cause less phases and thus, a less good fit. For the finally chosen initial values the algorithm yields the phases specified in table 2.3. We note the increasing rate, which is obvious, because we fit the phases starting with the slowest.

Table 2.3: H_4 phases fitted to a sample drawn from the power law distribution $P_L(2,1)$

phase	1^{st}	2^{nd}	3^{rd}	4^{th}
α_i	0.05400	0.30988	0.49589	0.14023
λ_i	0.18310	1.14088	2.65827	10.1389

Helpful in deciding the initial values are the splitting factors: If the first is high, we likely have chosen a too small starting point and should increase c_1. If the last is high, the matching points decline too slowly such that the procedure does not reach the lower part of the distribution. In that case we either should increase d or use more phases, meaning to choose a bigger k.

Finally, note that for $b \to 1$ the λ_i in equation 2.84 becomes

$$\lambda_i = -\frac{\mathrm{d}}{\mathrm{d}t} \ln\left(F^c(t)\right)\big|_{t=c_i} = \frac{f(c_i)}{F^c(c_i)} = r(c_i), \tag{2.87}$$

the so called *hazard rate function* known from failure distributions (survival theory), which we use in section 2.3.1 figure 2.27.b to graphically compare distributions.

Cox_k fitting scheme

Using H_k we cannot achieve models for smooth processes. For these we need chains of Markovian phases, which the *Cox model* $Cox_k(\vec{\lambda}, \vec{p})$ (figure 2.32) provides most flexibly. Actually, the Cox model can be used for both, smooth and bursty processes. The fitting approaches presented in [34,

chapter 7.6.7] are different for the two cases and briefly outlined here presenting condensed equations using the herein consistently chosen nomenclature (different from sources).

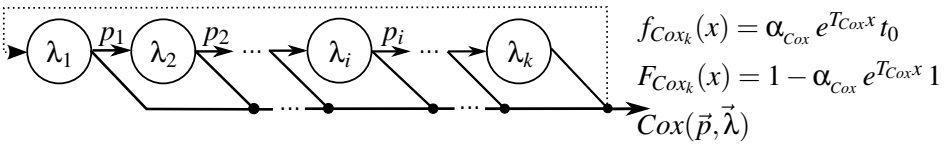


Figure 2.32: The Cox model

To model a process with a *coefficient of variation* $c_X \leq 1$ we reduce the number of parameters by setting $\lambda_i = \lambda$ and $p_i = p$. This converts the Cox into the simpler geometric mixed Erlang process $E_{k,p,\lambda}$, which still is sufficient to precisely match the first two moments for integer k. A feature that cannot be achieved using the plain Erlang-k process. Contrary to the general *Cox* distribution the *pdf* and *cdf* of $E_{k,p,\lambda}$ can be stated based on the geometrically weighted summation of the different *Erlang-k* distributions it is composed of.

$$f_{E_{k,p,\lambda}}(x) = (1-p)\sum_{i=1}^{k-1} p^{i-1} f_{E_i}(x) + p^{k-1} f_{E_k(x)}$$

$$F_{E_{k,p,\lambda}}(x) = (1-p)\sum_{i=1}^{k-1} p^{i-1} F_{E_i}(x) + p^{k-1} F_{E_k(x)}$$

The equations for the mean value and the variance of $E_{k,p,\lambda}$ have been derived in [34] using Laplace transformation based on $F_{E_{k,p,\lambda}}(s) = (1-p)\frac{\lambda}{s+\lambda} + p\prod_{i=1}^{k}\frac{\lambda}{s+\lambda}$

$$\mathrm{E}\big[E_{k,p,\lambda}\big] = \frac{1+p(k-1)}{\lambda}, \qquad Var(E_{k,p,\lambda}) = \frac{1+p(k-1)+p(1-p)(k-1)^2}{\lambda^2} \tag{2.88}$$

from which the equation for the square coefficient of variation results.

$$c^2_{E_{k,p,\lambda}} = \frac{Var(E_{k,p,\lambda})}{\mathrm{E}\big[E_{k,p,\lambda}\big]^2} = \frac{1+p(k-1)+p(1-p)(k-1)^2}{(1+p(k-1))^2} \tag{2.89}$$

Choosing $k = \lceil \frac{1}{c^2_X} \rceil$ this equation is used to calculate the tapping factor p, for example as recommended in [55], to finally get the rate λ for the k phases.

$$p = \frac{k - 2c_X^2 + \sqrt{k^2 - 4kc_X^2 + 4}}{2(c_X^2+1)(k-1)}, \qquad \lambda = \frac{1+p(k-1)}{\mathrm{E}[X]} \tag{2.90}$$

If we use the above equations to fit the $E_{k,p,\lambda}$ distribution to the repeatedly used example, we get $k = 8$, $p = 0.9815$, and $\lambda = 3.2073$. This result is depicted in figure 2.33. The solution is very close to the result we achieved when fitting the Erlang-7 distribution. Primarily due to the high p, which causes that the entire phases-chain dominates the output process. However, while with E_k the curves become steeper with increasing k, here the tapping causes that for small x the $E_{k,p,\lambda}$ is above E_7 although it uses more phases. For large x the two become identical, despite the different number of phases. If we use the above equations to fit the $E_{k,p,\lambda}$ distribution to the uniform distribution $U(1,4)$, we get $k = 9$, $p = 0.9901$, and $\lambda = 3.5683$. The result is depicted in figure 2.34. Clearly, this is not a perfect fit. However, looking at the *cdf*s we recognise that the matching is not too bad.

To model a process with a *coefficient of variation* $c_X > 1$ the number of parameters is reduced by pre-selecting $k = 2$, restricting the model to two phases. This leaves three open parameters, λ_1, λ_2,

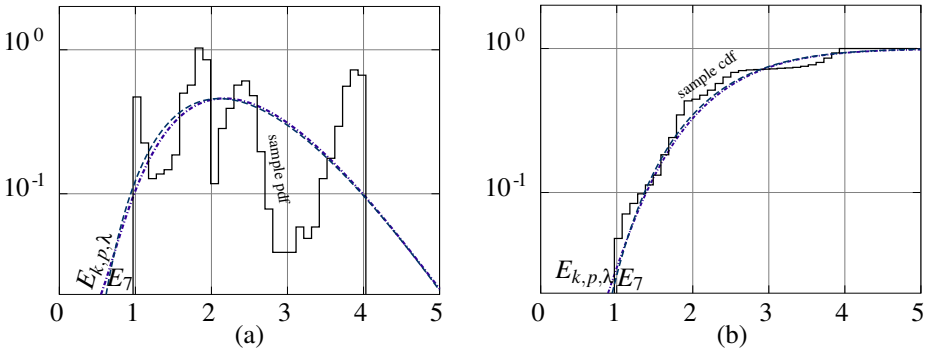


Figure 2.33: $E_{k,p,\lambda}$ (slash-dotted) fitted to the example, (a) *pdf*s, (b) *cdf*

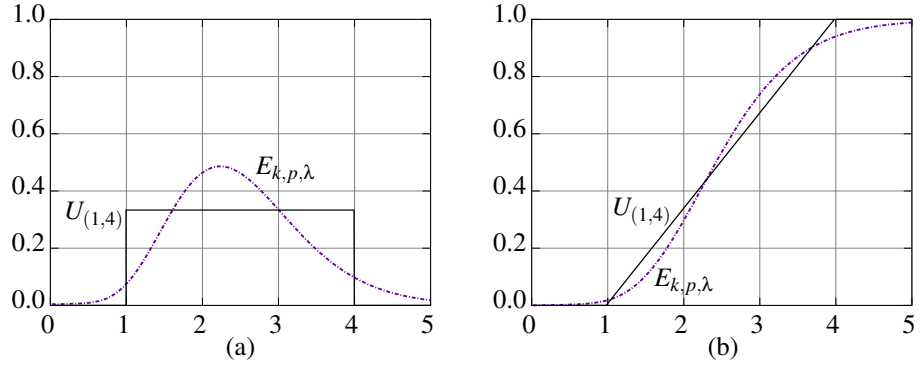
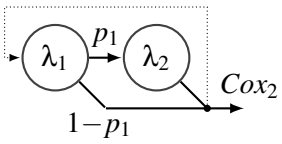


Figure 2.34: $E_{k,p,\lambda}$ (slash-dotted) fitted to the uniform distribution $U(1,4)$, (a) *pdf*s, (b) *cdf*s

and the piping factor p_1 in between the two phases. Again, three parameters are more than sufficient to perfectly match the first two moments.

The same procedure as before, given in [34], is used to get matching parameters.



$$\mathrm{E}[Cox_2] = \sum_{i=1}^{k} \frac{\prod_{j=1}^{i-1} p_j}{\lambda_i} = \frac{1}{\lambda_1} + \frac{p_1}{\lambda_2} \tag{2.91}$$

$$Var(Cox_2) = \frac{1}{\lambda_1^2} + \frac{p_1(2-p_1)}{\lambda_2^2} \tag{2.92}$$

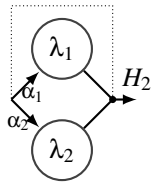
$$c^2_{Cox_2} = \frac{\lambda_2^2 + \lambda_1^2 p_1(2-p_1)}{(\lambda_2 + p_1\lambda_1)^2} \tag{2.93}$$

Out of the infinitely many possible solutions, a very convenient one is recommended in [55],

$$p_1 = \frac{1}{2c_X^2}, \qquad \lambda_1 = \frac{2}{\mathrm{E}[X]}, \qquad \lambda_2 = \frac{1}{c_X^2\,\mathrm{E}[X]} = p_1\lambda_1 \tag{2.94}$$

which actually can be used for any $c_X^2 \geq 0.5$, not only for $c_X > 1$.

Given that $c_X \geq 1$ latter can be achieved similarly by a H_2 model. In particular, we get



$$\alpha_1 = \frac{1}{2}\left(1 + \sqrt{\frac{c_X^2 - 1}{c_X^2 + 1}}\right), \quad \lambda_1 = \frac{2\alpha_1}{\mathrm{E}[X]}, \quad \alpha_2 = 1 - \alpha_1, \quad \lambda_2 = \frac{2\alpha_2}{\mathrm{E}[X]} \tag{2.95}$$

if we apply the balancing condition $\frac{\alpha_i}{\lambda_i} = const$ common with H_k models.

2.4 Traffic flows and network load composition

Data exchanged over some time span is called *traffic flow*. The network entities that may exchange data (*communicate* with each other) have to be peers in the OSI-model, detailed in table 2.4 as standardised in ITU-T X.200 [1]. Peers reside on the same layer. Thus, any interface between

Table 2.4: The OSI-model — functional layering enables integration of different implementations

layer : name	prime task	implementation examples[1]
7 : *application* layer	provide service	http, smtp, ftp, nfs, ntp, snmp, sip
6 : *presentation* layer	adjust data formats	MIME, SSL, TLS, encrypt, compress
5 : *session* layer	coordinate flows	RTP, CoS, scheduling, streaming
4 : *transport* layer	control data flow	UDP, TCP, SCTP, DCCP
3 : *network* layer	establish connectivity	IP, OSPF, IS-IS, CONS
2 : *link* layer	transfer data units	LLC, HDLC, SDLC, PPP, MAC
1 : *physical* layer	maintain bit-streams	TX/RX, CRC, encoding, interleaving
0 : *physical medium*	propagate analogue signals	OOK, PSK, QAM, OFDM, FHSS

two layers can be used to identify a traffic flow. Commonly, it is defined among layer 7 entities, layer 4/3 interfaces, or layer 3/2 interfaces. Below layer 2 we usually call it a bit-stream on layer 1 and symbol-sequence on layer 0 because here it is no more recognised as identifiable data. The information has become a digital, and finally analogous signal that can actually propagate across a physical medium and thereby leave its current location.

The duration (*lifetime*) of the virtual communication links (*connections*) used to exchange data may be unknown until it actually ends. The load distribution across the lifetime of a connection, meaning within a flow, is commonly assumed to be independent and identically distributed over the entire lifetime of a connection, but it is not necessarily the same for both directions. Summarised this means that

- the demand to communicate defines the connection that enables peers to exchange data,
- the communication reason and content cause the flows lifetime distribution and volume,
- the data exchange mechanism applied determines the flows internal load distribution,

and, that load and flows can be specified per layer of the OSI-model. Flows and loads on lower layers are in general the result of flows and loads on the layers above. Layer specific transport mechanisms may howsoever change the internal properties, being the size- and time-distribution of load units.

Short-lived flows are sometimes called bursts, and only long-lasting communication sessions are referred to as flows. The lifetime is a fundamental parameter, which may be distributed in any form. Thus, it does per se not qualify for separating bursts from flows. We use a different criterion: a burst is defined as traffic load that occupies the available bandwidth without gaps for a certain time period. Consequently is a burst an atomic load unit, comparable with a packet or frame. However, load units may be split or combined when they pass through different layers. Thus, the definition of load units is layer specific. Because flows and queueing systems can be found on any layer, we do not distinguish the terms used to name different load units, but use them interchangeably, if not otherwise mentioned.

In addition to the data exchanged by applications (the *payload*) also network components exchange data for control and management issues (the *overhead*), which also contributes to traffic flows. This internal communication is either performed by independently transported signalling messages, or is encapsulated in data unit *headers*, which anyhow may become attached when data units pass layer interfaces.

[1] Intentionally protocol acronyms are here not explained, please see the literature if not readily known.

2.4.1 Network load definition

A network is a macro entity: it provides the connectivity among the resources required to transport traffic flows from one interface (the *source*) to the intended other interface (the *destination*). The network elements from all OSI-layers (table 2.4) build together a network; no part or layer is a network, only together they become a network. The layer 3, called network layer, manages the connectivity. It does not transfer data units, that is performed by layer 2 components. Nor does it prepare, manage, or control the data exchange, these tasks are carried out on layers 6, 5, and 4.

To visualise what *network load* may be we use the *stock-flow* model depicted in figure 2.35. The

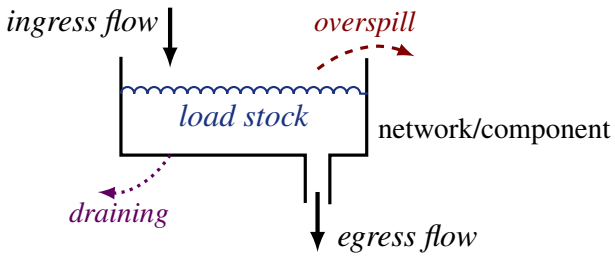


Figure 2.35: The stock-flow model used to explain the term network load

model abstractly relates the factors that are of importance. First, the *ingress* load, being the load that flows into the model. Second, the amount of load stored within the model, which defines the amount of load received but not yet delivered, the *stock*, which we call *backload*. Third, the *egress* load, being the load flow that leaves the model, also called *throughput*, although this term may also refer to its maximum. Finally, *overspill* and *draining* identifying load shares that enter the model but never become delivered, which we call *loss* in its total.

What is network load? This question cannot be answered in general, the stock-flow model would suggest it is the backload. From the literature we recognise that this term is used ambivalently for:

1. the sum over all *offered ingress load flows* across all ingress points,
2. the *ingress load vector* containing the current load at every ingress point,
3. the *load vector* containing the current load flow into every network resource, or
4. some *artificial metric* that provides an average resource loading factor.

In this view, *network load* may be specified en gross or in detail, absolute or averaged. All these definitions apply to the stock-flow model, but none refers to the backload. The ambiguous definitions result from different views. The first two are top down, and define the network load as seen from the layer above, being layer 4, the transport layer. This view is very handy, but it lacks information on the distribution of the load across resources. The latter two are bottom up, and state the network load via the loading of the resources, being the basic components any network is constructed of: *nodes* and *links*. The network in total, as well as the individual resources, can be understood as stock-flow model: the network as dubious cloud that somehow delivers the inserted load to where it is destined for, the nodes as the layer 3 components that direct the load, and the links as the layer 2 components that transfer the directed load among neighbouring nodes.

Which definition shall we use? To identify the implications of the above listed definitions we define $\rho=1$ as the boundary between normal operation and overload. In this respect we recognise that the first definition makes sense if we can state the network's transport capacity. This is possible for some regular topologies with a unique bottleneck capacity, for example a ring, torus, or hypercube topology with static, ingress related routing. We do not use this definition because we do not intend to restrict our studies to specific network designs. The second definition is more general and comprises sufficient information to derive the next definition if the routing of the contained ingress loads is known. Thus, this definition will be used to state examples. However, we call it *ingress load*, because

for a precise performance evaluation the distribution across resources is commonly required, which is the meaning of network load as specified by the third definition.

The third definition is the most accurate and it is always applicable, even for distributed dynamic routing. This is the meaning of the term *network load* we henceforth refer to, if not otherwise mentioned. Note, the $\vec{\rho}$ vector contains the load *offered* to network resources, not the transported (*carried*) loads, which constitute the *utilization vector* $\vec{u}$. These vectors are related: the sum over the transported loads of preceding resources directed to a resource, plus the ingress load entering the network at a resource, is the load offered to succeeding resources

$$\rho(t^+) = \alpha + R \cdot u(t^-), \tag{2.96}$$

where R, the *routing matrix*, and α, the *ingress load vector*, are assumed to be time invariant. The time index t is used to express the time lag, meaning that t^- is an infinitesimal moment prior t^+. If no losses can occur, the offered and the transported loads are identical in their mean values, $\mathrm{E}[\rho(t)] = \mathrm{E}[u(t)]$. Else, a difference is caused by the *loss vector* δ ($u = \rho - \delta$). The mean loss vector and utilization components result from

$$\delta(j) = p_l(j) \cdot \rho(j), \qquad u(j) = \big(1 - p_l(j)\big) \cdot \rho(j), \tag{2.97}$$

where $p_l(j)$ is the loss probability at network element j, which we also assume to be time invariant. Considering how these vectors cross-determine their change over infinitesimal time-steps, we recognise the common iterative approach to solving queueing networks. Figure 2.36 depicts a small network of five nodes. Every node handles the flow aggregate that enters it, and every link

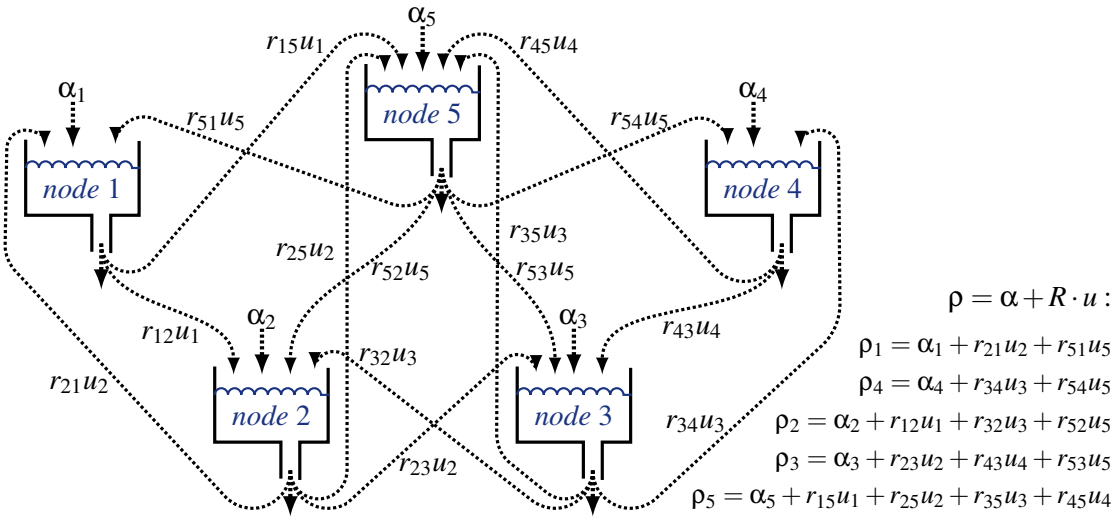


Figure 2.36: Flow exchange and resultant load distribution among network resources

forwards the composite of all the flows that use it. Consequently depends the load of any node somehow on the utilization of every node. To solve the interdependences an equation system is set up. An essential part therein is the correct merging of the traffic flows that enter a resource. The analytic flow aggregation by distribution composition is discussed in the next subsection.

The forth network load definition $\bar{\rho}=\mathrm{E}[\rho_j]$ is useful to assess the performance of network management and control. We extend it by $max\{\rho_j\}$ and $Var(\rho_j)$, at and across resources, to study the performance with a scope exceeding the plain mean-of-means analysis. Specific implications are discussed with the examples presented in later chapters, where these metrics are used to evaluate the qualities of traffic management and control mechanisms.

2.4.2 Summation of flows toward traffic aggregates

Given the rules of mean values presented in section 1.5 we can calculate the mean of the sum of different distributed flows quite simply by weighted summation: $\mathrm{E}[X_\Sigma] = \sum u_i \mathrm{E}[X_i]$. If we know the individual distributions *pdf* $f_i(x)$ or *cdf* $F_i(x)$, and may assume that all flows are independent, than we can directly calculate their weighted sums likewise.

$$pdf: \quad f_{X_\Sigma}(x) = \sum_{i=1}^{n} u_i f_i(x), \qquad cdf: \quad F_{X_\Sigma}(x) = \sum_{i=1}^{n} u_i F_i(x) \tag{2.98}$$

Apparently simple, the result can be a very unhandy distribution function, in particular if the flows individual peak probabilities are differently located. The resultant distribution may become a so called *multi-modal* distribution, which is characterised by having several peaks at different x values. This problem does not occur if we model all flows by strictly declining distributions with their only peak at zero, for example by modelling all using a negative exponential or *Lomax* distribution.

Until here there was no need to separately discuss service time distribution and inter-arrival time distribution. To merge traffic flows, which are jointly defined by a service time distribution and an inter-arrival time distribution, we have to consider some implicit differences. Thus, we split the merging topic, and show separately how to get the according aggregate distributions.

Service time distribution

The service time is the duration that a load unit occupies a resource. It is therefore also called *holding time* T_h. Considering the *stock-flow model*, it is the time some amount of load requires to pass the exit port. In terms of queueing systems it is the time the server needs to serve a customer, where the load unit is the customer and the network resource is the server.

Most physical layer resources provide bit-transfer at constant bit-rates. However, the service times T_h are not deterministic due to variable sized load units (packet-/frame-lengths X_i), and the commonly applied link control protocol, which may trigger re-transmissions in case of failed transmissions. The transfer on layer 0 happens at the speed of light. The holding time per bit on layer 1, being the time required to transmit a single bit, is a resource specific constant equal the inverse *line-rate*. This is not the speed of the signal carrying the bits, it is the speed at which the resource can modulate the carrier signal, being the speed at which bits can be put on the line. The layer 2 speed, considering the reduction caused by the link control mechanisms, is the resource's *capacity* c_j, where j identifies the resource. Actually, this is not a constant. However, the common control mechanisms roughly preserve continuity, and thus c_j may be considered constant in average. Therefore, we can calculate the holding time T_{ij} and the *mean service rate* μ_{ij} for flow i at resource j by

$$T_{h,ij}\,[\mathrm{s}] = \frac{X_i\,[\mathrm{bit}]}{c_j\,[\mathrm{bit/s}]} = \frac{X_i}{c_j}\,[\mathrm{s}], \qquad \mu_{ij} = \frac{1}{\mathrm{E}\left[T_{h,ij}\right]} = \frac{c_j}{\mathrm{E}[X_i]}\,[\mathrm{unit/s}]. \tag{2.99}$$

If the capacity cannot be assumed to be constant, we either need to replace c_j by $c_j(t)$ or model it distributed as random variable C_j. For queueing systems it is common to define the time unit such that μ=1 is achieved. This simplifies equations, but cannot be performed where different $T_{h,j}$ occur.

If the traffic that passes a resource is the aggregate of different flows, we need to merge the different service time distributions into one in order to achieve a joint model. For a server that serves load units randomly one after the other we can assume independence and apply equations 2.98 to calculate the service time distribution of the aggregate. If only the first two moments are defined, we get the moments of the aggregate from

$$\mathrm{E}\left[T_{h,j}\right] = \frac{1}{c_j}\sum_{i=1}^{n} u_i \mathrm{E}[X_i], \qquad \mu_j = \frac{1}{\sum\limits_{i=1}^{n} \frac{u_i}{\mu_{ij}}}, \qquad Var(T_{h,j}) = \frac{1}{c_j^2}\sum_{i=1}^{n} u_i^2 Var(X_i). \tag{2.100}$$

Equations 2.98 and 2.100 are commonly applicable if all arriving load units become served. If losses occur independent and identically distributed these only affect the u_i and the equations are still applicable. However, losses are commonly load dependent, which causes that this simple calculus does not yield precise results. In case of controlled transmission the entire load unit may become re-transmitted if it has not been successfully transmitted in the first attempt. This adds a fraction composed of the response time lag and the repeated transmission time, weighted by the transmission failure probability, to the average time the server is occupied by a load unit. Evidently, this increases the transmission time variance heavily. Similarly, on error prune radio links, the layer 2 transmission control causes high variability, which should be considered in a similar way.

Inter-arrival time distribution

The inter-arrival time T_a is the time-span in between the arrivals of two subsequently arriving load units. Along a serial transmission link, no load unit can arrive prior the previous has completely arrived, meaning that before the next can arrive, the last bit of the previous must have been received. This fact is widely neglected because at low load levels the probability for overlapping arrivals is marginal. For negative exponentially distributed arrivals it can be shown that the special properties of the negative exponential distribution equalise the effect. However, real world traffic is not negative exponentially distributed, and in the literature this has been discussed vividly. The term related to this issue is *streamlining effect* [56]. The issue is mentioned for awareness only, in general we comply with the common practice to ignore it, accepting that all calculations become approximations.

To describe the aggregate of individual flows it is more convenient to use the inverse of the inter-arrival time, the *arrival rate* $\lambda_j = \frac{1}{\mathrm{E}[T_{a,j}]}$. Merging represents a summation in terms of mean arrival rates,

$$\lambda_j = \alpha_j + \sum_{i=1, i\neq j}^{n} u_i(j^-), \qquad \mathrm{E}[T_{a,j}] = \mathrm{E}[T_{a,0j}] + \frac{1}{\sum_{i=1, i\neq j}^{n} \frac{u_i(j^-)}{\mathrm{E}[T_{a,i}]}}, \tag{2.101}$$

where the $u_i(j^-)$ state the remaining offered intensities of the contributing flows at resource j, being the carried intensity u_i at the previous resource j^-. This simple approach is only applicable for mean values, because the arrival distribution becomes a departure distribution $f_{D,ij}(x)$ whenever passing a resource, which is load dependent, contrary to a flow's service time distribution, which generally does not change hop by hop.

To determine an aggregate's arrival distribution shape it is necessary to calculate all contributing departure distributions $f_{D,ij^-}(x)$ of preceding resources. If this is possible, we can use equation 2.98 to merge the contributing rate distributions

$$\frac{1}{f_{A,j}(x)} = \frac{1}{f_{A,0j}(x)} + \sum_{i=1, i\neq j}^{n} \frac{u_i(j^-)}{f_{D,ij^-}(x)} \quad \text{or} \quad \frac{1}{F_{A,j}(x)} = \frac{1}{F_{A,0j}(x)} + \sum_{i=1, i\neq j}^{n} \frac{u_i(j^-)}{F_{D,ij^-i}(x)}, \tag{2.102}$$

where $f_{A,0j}(x)$ and $F_{A,0j}(x)$ are the distribution of the flow entering the network at resource j with intensity α_j. However, deriving the $f_{D,ij^-}(x)$ or $F_{D,ij^-}(x)$ is commonly intractable due to the non-linear influence of the queueing process. Particularly, the cyclic dependences are hard to solve in case of meshed networks. More practical are rough assumptions based on the central limit theorem, solid guessing, and experience. After few iterations a feasible guess should in general be achieved. Alternatively, measured shapes may be tuned to fit the calculated first moments. Both approaches demand cognitive decisions upon the applied simplification and yield approximates only.

A more methodical approach is shown in [57]. For assumed renewal property an equation to calculate the squared coefficient of arrival variation $c_A^2(j)$ can be derived based on [58]

$$c_A^2(j) = \sum_{i=1}^{n} \frac{u_i(j^-)}{\lambda_j} c_{D(i)}^2(j^-) = \frac{\alpha_j}{\lambda_j} c_{A(j)}^2 + \sum_{i=1,\, i\neq j}^{n} \frac{u_i(j^-)}{\lambda_j} c_{D(i)}^2(j^-), \tag{2.103}$$

where $c^2_{D(i)}(j^-)$ is the squared coefficient of departure variation of flow i when leaving the preceding resource j^-. The latter equation separately states the ingress flow entering the network at resource j, as it is in the end required to define the system of equations. Assuming that the per flow departure characteristic is related to the aggregate characteristic only, we may continue with the approach outlined in [57] and use the splitting proposed under the assumption that the departure process would be of renewal type to get an equation for the departure characteristics

$$c^2_{D(i)}(j^-) = u_i(j^-)\, c^2_D(j^-) + (1 - u_i(j^-)). \tag{2.104}$$

For the aggregate departure characteristic $c^2_D(j^-)$ a nice approximation is derived in [57]

$$c^2_D(j^-) = u(j^-)^2\, c^2_B(j^-) + (1 - u(j^-)^2)\, c^2_A(j^-), \tag{2.105}$$

where $u(j^-)$ is the utilization of the resource j^-, and $c^2_B(j^-)$ is the aggregate coefficient of service time variation. It depends on the aggregate arrival characteristic $c^2_A(j^-)$ entering the preceding resource j^-, which needs to be calculated likewise based on the flows entering it. This proceeds across the entire network of resources, defining a system of equations that is linear in terms of squared coefficients of arrival and service variations c^2_A and c^2_B,

$$c^2_A(j) = \frac{\alpha_j}{\lambda_j} c^2_{A(j)} + \sum_{i=1, i\neq j}^{n} \frac{u_i(j^-)}{\lambda_j} \left(u_i(j^-) \left(u(j^-)^2\, c^2_B(j^-) + (1 - u(j^-)^2)\, c^2_A(j^-) \right) + (1 - u_i(j^-)) \right)$$

that can be solved rather easily, once we solved 2.101 to get the load aggregates $u_i(j)$ at each resource j for every flow i present. To do so, we need given ingress flow intensities α_i, flow routing across resources, being j sequences per flow i, and the service time characteristics $c^2_B(j)$, which may be composed from the present flows according to 2.100, adding one more dimension.

In general equations 2.103, 2.104, and 2.105 are approximations only because all contributing flows have to be independent and of renewal type to make this approach precise. The first is not necessarily true, for example if two flows pass a common preceding resource, and the second can be assured only if all initial flows result from Poisson processes and all service times across all resources are negative exponentially distributed. This is the case for *Jackson networks* only [59], but not in general. Consequently, the proposed approach approximates the second moment solution only.

Finally note, in general the results calculated for a traffic aggregate cannot be decomposed into performance metrics per contributing flow. To evaluate systems on a per flow level, it is possible to use *vacation models*, where the serving process becomes split among different traffic flows. Per flow a sub-model is designed, where the serving is interrupted while load units from other flows are served. This is a contrary approach because the individual service time distribution is preserved, which is possible because with non-preemptive serving only the waiting time prior service depends on the properties of the other flows currently present. Vacation models are widely discussed in [60], it was used exemplarily in section 1.4.3 (figure 1.20), and its application is briefly discussed in section 4.1.2 on strict prioritisation systems.

2.4.3 Typical traffic flows

The characteristics of traffic flows are determined the specifics of the used applications. However, several mechanisms in and between layers influence the distribution of load units on different layers. Due to the plurality of applications and mechanisms, their implementation diversity, and the ever continuing modification of the immanent protocols and mechanisms, we only define basic types here. Real traffic flows will in general differ from the here defined *archetypes*. To study particular applications and mechanisms considering implementation specifics the traffic flows should be defined more precisely. For example, via recorded traces applying the tools presented in section 2.3.

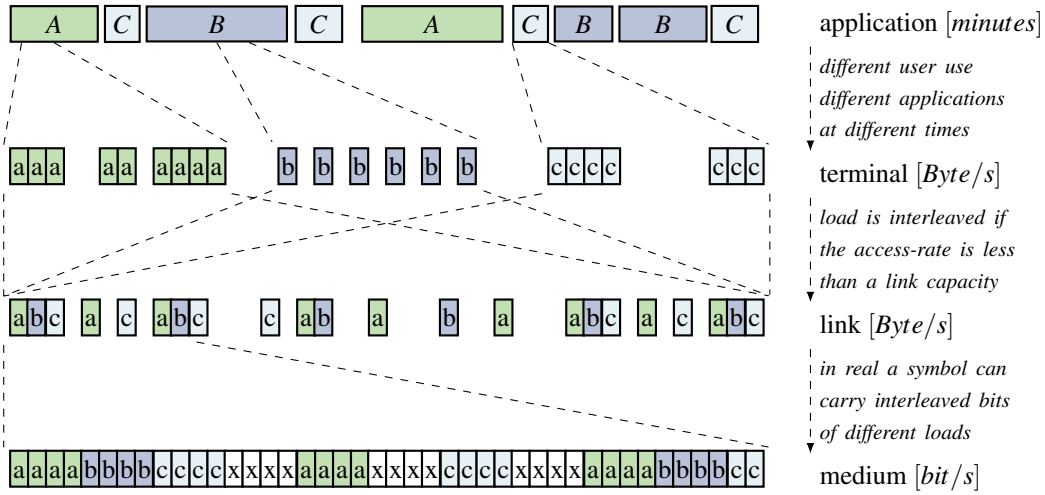


Figure 2.37: Hierarchic traffic load aggregation

Different layer specific load archetypes are sketched from deductive reasoning based on application archetypes, typical layer mechanisms, and the aggregation hierarchy depicted in figure 2.37. It is based on [61], extended by application and physical layer as well as the aggregation of competing flows. For clarity only we show constant packet size and no interleaving at the physical layer.

Application layer related load generation archetypes

A priori we state that the actions of human beings are in average random and uncorrelated. Correlation of human actions, particularly of cognitive decisions including the popularity of applications, is the core topic of the *game theory* and only relevant on time intervals much longer than what we consider here. Thus, on the here relevant time scale the average application usage by humans can be assumed independent and identically distributed concerning the usage profile. Therefore, the use of applications by humans, layer 8 in the OSI-model, can be modelled by a Poisson process, parametrised by the usage intensity only. Note, this assumption relates to the duration an application is used and the time in between subsequent usage of the same as well as different applications. This does not necessarily cause the traffic to be negative exponentially distributed.

- **Markovian profile:** If per application negative exponentially distributed loads are caused, a Poissonian traffic flow results. Typical example applications are *file transfer*, *e-mail exchange*, *Internet surfing*, and *video on demand*, because the initiation times and the transmitted load can be assumed to be negative exponentially distributed.
- **Modulated profile:** If an application constantly causes usage independent low traffic load and load peaks occur on user demand, than it is best modelled as *Markov modulated Poisson process* (MMPP). Typical examples are sensor controlled *surveillance* and *monitoring* services, including automated e-mails in case certain events occur. The load size caused by these applications is typically quite smooth, if not constant.
- **Persistent profile:** Persistent services with no user interaction cause machine driven data flows. These are rather deterministically in both, timing (clocked) and data amount (message size). Still, to cover some variability in both, we model the traffic caused either by a narrow *Beta distributed* or a high order *Erlang distribution*. This type covers *machine-to-machine* applications as well as *broadcasting* services.

In their primary property these three archetypes represent Markovian, bursty, and smoothly distributed load generation intervals, in the order they are defined above.

Transport layer related flow composites

On the transport layer the load generated by applications is present in the form of transportable load units, for example IP-packets. The load caused is therefore defined as (average) *packets per second*. The normalisation to an average load unit size makes the approach more generally applicable. Still, we have to consider the properties of the load units that can be transported. Evidently, their size can neither be zero nor infinite. Considering IP, the IP-header states a lower bound, and the maximum IP-packet size of 1.5 kByte an upper bound. Normalising to an average size of 250 Byte an approximate size range from 0.1 to 6 results. The *ingress* rate for IP-packet arrivals is limited by the processing power at both terminals, which we assume infinite. The maximum *egress* rate refers to the packet reception and is limited by the bottleneck capacity, which should be part of the model.

Many applications do not occupy the entire provided capacity all the time they are active. Thus the application related transport layer load results from modulating the load generation defined above with the packet distribution caused during use. The resultant distribution could be mathematically derived, but for modelling less complex distributions are advantageous.

- **Bursty loads:** Every file transfer defines a burst load because files are in general much bigger then a single transport unit. The arrival of a file to be transported causes a burst of packets to be sent as soon as possible. In case of UDP each file causes a *correlated load burst*, where the inter-arrival time equals the packet size until all fragments of the file have been put on the go. Considering the influence of the sending rate control of TCP, we get *smoothly distributed load bursts*, where the times between bursts and the size of bursts are determined by the round-trip-time and the current transport-window-size.
- **Heavy tailed loads:** The idle intervals in between bursts shrink exponentially if bursty traffic flows become merged. In general the resultant aggregate load is not Markovian. Today, this traffic type is assumed to be heavy tailed, and modelled by a *power law distribution*, for example Pareto or Lomax. However, as explained in subsection 2.4.2, the service time distribution of the composite is the weighted sum over those contributing, and this is more smooth than that of a single user flow because each is itself bounded by $[0.1\,..\,6.0]$.
- **Smooth loads:** Live communication lasts long compared to average file transfers, and commonly causes low average loads compared to the transport capacity. On layer 4 we can assume such services as infinitely lasting. The amount of data required to maintain a lasting communication is in the long term rather constant. In case of simplex communication, for example web-radio and live streaming, the resultant flow can be modelled by a narrow *Beta* or a high order *Erlang* distribution in both, inter-arrival times and packet size.
- **Markovian loads:** In case of halve-duplex communication, where one side is idle while the other transmits a constant flow of information, for example idealised VoIP, the inter-arrival time becomes a Markovian interrupted process. The interruptions increase the variance of the primarily smooth process. If idle and busy durations are per se negative exponentially distributed and in average equal long, we may assume the resultant inter-arrival times to be negative exponentially distributed. If variable data reduction and compression mechanisms are applied, also the size distribution may approach the negative exponential distribution.

In addition to the application driven loads we have to consider also the network internal traffic load required to perform the signalling among upper layer protocols and mechanisms.

- **Signalling loads:** Signalling traffic is commonly not negative exponentially distributed in either aspect because protocols commonly use deterministically sized signalling messages. These messages typically cause a chain of messages. Consequently is the inter-arrival time of *signalling traffic* best modelled by a *Markov modulated Poisson process* (MMPP) with rare high load intervals and long lasting low load intervals. The packet size can be modelled as before by a narrow *Beta* or a high order *Erlang* distribution.

Link layer related traffic aggregates

The load units on the link layer are the same as on layer 4, and we assume the same load size normalisation. If a unique, time invariant mean size exists we may also normalise the time to 1 time unit, because the link defines a constant capacity (Byte/s) and thereby the mean holding time, being the mean unit size divided by the link capacity.

Relating the traffic on this layer to the applications that may contribute to an aggregate is cumbersome: too many variables and mechanisms interfere. Traffic aggregates composed in that detail would never be general because hardly ever will all components be identically present in practice. Therefore, we sketch archetypes based on the link's location in the network.

- **Access traffic:** Access traffic results directly from the transported loads defined above because the few devices commonly connected to a modem cause traffic alike application in parallel. This generally applies for most local area networks (LANs). Considering that most arrivals appear in smoothly distributed bursts, we either get a smooth inter-arrival time process with negative exponentially distributed load batches, or harmonise it into a frequently interrupted negative exponential distribution, being the *Interrupted Poisson process* (IPP). To consider long lasting no-use intervals, which heavily depend on day-time and week-day, we need to add another interruption process. A H_3 should be sufficient to model the access inter-arrival time distribution for any long term load fluctuation. Concerning the size distribution we may use a rather flat *Beta* distribution or a low order *Erlang* distribution. This yields a $H_3/E_2/1$ queueing system. If we incorporate signalling and control traffic with independent and identically distributed small load units, a $H_2/M/1$ queueing system may be sufficient.
- **Metro traffic:** Metro traffic comprises load distribution in the down-link (service provider to user) and load concentration in the up-link (user to service provider). It can be modelled as merge of access traffic flows. If we assume all access traffic to be independent and identically distributed, the merged inter-arrival time distribution will show exponentially less lasting idle times, depending on the number of access traffic flows routed over a link. If also the size distribution is independent and identically distributed, the size distribution of the aggregate is the same. Consequently, the inter-arrival time becomes less bursty the more access flows are merged, and thus, a $H_2/E_2/1$ model should be sufficient. If we again incorporate signalling and control, an $M/M/1$ queueing system may be sufficient.
- **Core traffic:** Finally, interruptions are hardly identifiable if thousands of flows are merged. Only the mean traffic intensity depends on the day-time and week-day, but hardly its distribution. The literature on these aggregates is ambiguous: some sources propose that it is smooth while others stress that it is fractal. To our understanding it should be rather smooth due to the many effects that balance the different characteristics of the contributing load types. However, a prove of this argument is missing.

In addition to the application driven traffic loads there is prioritized signalling traffic present as well. The packet size used for signalling is protocol specific, close to the minimum packet size, and best modelled by a narrow *Beta* or high order *Erlang* distribution.

- **Signalling traffic:** On this layer the signalling traffic occurrences directly with the traffic it relates to. Thus, the inter-arrival time of *signalling messages* is best modelled by a *Markov modulated Poisson process* (MMPP) with short high load intervals. However, we have to add layer 3 signalling as well, which splits in on-demand messages to be incorporated here, and background processes that cause *control traffic*.
- **Control traffic:** Control traffic represents the persistent information exchange among network entities that is required to keep a network operational. This comprises information polling as well as scheduled information exchange, for example autonomous neighbour detection. Several protocols operate in parallel, and we can assume that the traffic caused is independent and identically distributed. Consequently, we can assume negative exponentially distributed inter-arrival times if the number of background processes is sufficiently high.

Note, header fields used for signalling cause no traffic contribution. These are readily considered by the overhead factor that increases the average load layer by layer. Only messaging that is transported alike regular traffic flows needs to be modelled by a specific flow type.

Physical layer loading

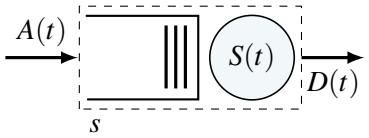
Finally, all way down we find the loading of the transmission medium, being in principle a pure *on-off pattern* with complex, load origin and aggregation dependent, autocorrelation among on-intervals. Of particular interest are the different resource sharing approaches, because these primarily define the behaviour and performance of the physical medium, which substantially influences the network performance in terms of worst case end-to-end delay and reliably achievable throughput.

- **Wired transmission:** Wired point-to-point transmission commonly relies on persistent transmission, where *idle patterns* are transmitted when nothing from the layer above can be transmitted. The physical layer never becomes idle, and no queueing occurs. Multiplexing schemes split the physical capacity into useful shares (channels), where each represents an independent resource. With time division multiplexing (TDM) the medium is not always instantly available. However, in average it is because a TDM channel is a $D/D/1$ queueing system, where the *backload* (queue filling) is always <1 because it depends on the phase difference between arrivals and departures only as long as $\lambda \leq \mu$. Such queues we call *buffers* because they never queue customers. They only solve timing issues by inserting delays less than a single slot duration. If we presume buffers, the physical medium can be modelled as $GI/G/n/n$ loss system, where the inter-arrival and holding time distributions are the merge of all contributions from the layer above, scaled to the capacity of the provided physical channels.
- **Shared transmission:** Wireless and other shared medium technologies transmit data on demand only. With uncoordinated sharing instantly upon arrival, lacking a joint control instance. This requires some *collision detection* and consequential *re-transmission* option, together with a *back-off* strategy that prevents repeated collisions. Physical layer transmission managed this way performs similar to the controlled transportation implemented in layer 4, although at a different time scale. They can be modelled as queueing systems with server vacations, or approximated as presented in section 4.1.
- **Slotted transmission:** The third option is distributed control, for example by a *token* scheme. In that case the access to the medium is persistently structured into *slots*, and the medium usage is advertised to all participating devices, such that no collisions may occur. A such shared medium can be modelled as multi server queueing system with server vacations. The structuring of the medium in (equally sized) shares defines n, the number of servers. The physical layer load distribution results from the distribution of the loads on the layer above: the upper layer inter-arrival time defines the inter-arrival times in between load batches, and the upper layer holding time defines the batch size distribution M because to transmit a load chunk we need m blocks that fit into one resource share, a single slot, each. The holding time on the slotted physical layer is commonly deterministic and equal the slot size. For access and local area networks, where such sharing is quite common, we get a $H_3^{E_2}/D/n_{vac}$ queueing model.

Modelling the physical resources yields their utilization. However, maximising that causes a conflict with the demand for quality. For performance reasons the medium utilization needs to be kept sufficiently below saturation. Actually, the utilization of physical resources tells nothing about efficiency: the less efficient the load is transported, the higher is the resource utilization. Only the relation of network load to network utilization assesses efficiency.

3 Queueing systems

Queueing systems consist of some queueing space, where load can be temporarily stored, and at least one device that sooner or later serves the arrived load. Such a system is primarily defined by an *arrival process* $A(t)$, a *service process* $S(t)$, the *system size* s (space to hold load), and the order in which load passes the system, comprising the *queueing-/scheduling-/serving-discipline*. To identify



$A/B/n/s/c/discipline(s)$

queueing systems in a uniform way we use the extended *Kendall notation*, where A refers to the distribution family of the arrival process $A(t)$, B to the distribution of the service process $S(t)$, n states the number of servers, s the system size, c the available load (if bounded), and the *discipline* states how the load is queued and how serving is scheduled. If not explicitly stated, s and c are infinite, and FIFO (first-in first-out) queueing also called FCFS (first come first served) is assumed.

These models are perfectly suited to evaluate the performance of systems that handle traffic flows characterised by an inter-arrival and a service time distribution. Because performance metrics should be time invariant, we are primarily interested in the steady state, particularly

$$\text{the steady state-probabilities} \qquad \pi_i = p(i) = \mathrm{P}[X{=}i]$$

expressing the probability that i load units are currently in the system, either waiting in the queue or currently being served. X is called the *system process*, and Q the *queueing process*, being a part of X. To complete it, T_f is the *flow-time process*, and T_w the *waiting-time process*. Transient analyses are necessary to evaluate how quickly a system returns to its steady performance once it left steady operation conditions, but this property commonly depends on the actual cause.

If we can calculate all state probabilities π_i, many properties of the queueing system become instantly apparent. For simplicity we use an index-triple that explains the states they refer to: i shall always be the number of customers in the system, j the number of customers currently queued, and k shall be used to further separate states where required. This definition of i and j indices yields that the relevant equations can be system independently expressed by

$$\text{mean system filling} \qquad \bar{x} = \mathrm{E}[X] = \sum i\pi_{ijk} \tag{3.1}$$

$$\text{mean queue filling} \qquad \bar{q} = \mathrm{E}[Q] = \sum j\pi_{ijk} \tag{3.2}$$

$$\text{mean flow time} \qquad \bar{f} = \mathrm{E}[T_f] = \tfrac{1}{\lambda}\,\mathrm{E}[X] = \tfrac{\bar{x}}{\lambda} \tag{3.3}$$

$$\text{mean waiting time} \qquad \bar{w} = \mathrm{E}[T_w] = \tfrac{1}{\lambda}\,\mathrm{E}[Q] = \tfrac{\bar{q}}{\lambda} \tag{3.4}$$

where we apply

$$\textit{Little's law} \quad L = \lambda F \qquad \bar{x} = \mathrm{E}[X] = \frac{\mathrm{E}[T_f]}{\mathrm{E}[A]} = \lambda\bar{f} \tag{3.5}$$

which is generally applicable for any queueing system. Note that no summation restrictions are required for the defined i and j only, and that the mean arrival rate is expressed by $\lambda = \frac{1}{\mathrm{E}[A]}$. The *bars* over variables highlighting that these refer to mean values are henceforth not explicitly shown.

3.1 Infinite queueing systems

Infinite queues are commonly assumed where a saturation state $\pi_{ijk}(i{=}s)$ does not exist, $s{=}\infty$, or its probability is negligible, $\pi_{sjk}{\approx}0$. In case of non-negligible π_{sjk} the performance of a finite system can be approximately assessed by truncation, meaning adjustment of the infinite system's performance by a posteriori considering the non-existing state probabilities π_{ijk} for $i{>}s$. We start with these somewhat unhandy queueing systems because they are the basis of the entire queueing theory [14, 62], and highlight that such systems can be solved analytically. The derivation of analytic solutions comprises the prime content of many textbooks on queueing systems, see for example [14, 34, 57, 62–64]. Here, selected approaches are sketch and discuss for the sake of completeness, consistent notation, and to introduce solving strategies.

The general birth-and-death process

Commonly we presume that the inter-arrival and service times do not depend on the system state, are independent and identically distributed over time, and uncorrelated. Allowing system state dependent mean rates for both, arrivals and service times, we relax this proposition and get the so called *birth-and-death process*. It describes a realisation independent *continuous time* system of one dimensionally connected Markov sates, depicted in figure 3.1, where the inter-arrival and service rate may in any way depend on the current system state, but not on how it has been reached (memoryless).

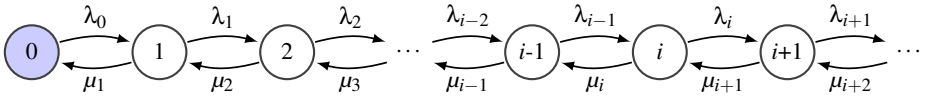


Figure 3.1: State transition diagram of a birth-and-death system

The *state transition diagram* models the system state process by the number of clients in the system i at any time, and its potential changes by transition rates, which equal the inverse transition times $r_{ij}{=}\frac{1}{\tau_{ij}}$. Independent of the actual rates, an arriving load increases the system state by one, a served load leaves the system decreasing the system state by one. Due to time continuity two events never occur at the precise same time instance and thus, no probabilistic splitting or state bypassing can occur. These are the properties that define a birth-and-death process.

In the *steady state* the probability of each state represents the probability that at a certain time the system is in that state, being $\pi_i{=}p(i){=}\mathrm{P}[X(t){=}i]$, while the duration that the system remains in a state once it is reached, called *state sojourn or transit time* τ_i, is determined by the exiting rates, including the exit of a currently served client but also the arrival of a new client, not to be mistaken with the *holding time h*, being the time required to serve a client ($h{=}\frac{1}{\mu}$). The fraction of time the system is in a state, which equals π_i, is determined by the frequency a state is reached times the duration the system remains in that state.

The birth-and-death process still represents a *homogeneous Markov chain*, where only transitions among neighbouring states exist and all the distributions are *time invariant*. The system is *memoryless*, and can be solved if we manage to include the precise rate dependences. However, even without specified rate dependences we can readily identify some generic properties.

The *differential-difference equations* of states are

$$\frac{\mathrm{d}p_i(t)}{\mathrm{d}t} = \lambda_{i-1}p_{i-1}(t) - (\lambda_i+\mu_i)p_i(t) + \mu_{i+1}p_{i+1}(t) \quad \forall_{i>0}, \quad \text{and} \quad \frac{\mathrm{d}p_0(t)}{\mathrm{d}t} = \mu_1 p_1(t) - \lambda_0 p_0(t), \tag{3.6}$$

and for the *steady state*, where $\frac{\mathrm{d}p_i(t)}{\mathrm{d}t} = 0$ needs to be fulfilled such that $p_i(t) = \pi_i \; \forall_t$, we get

$$\pi_{i+1} = \frac{\lambda_i+\mu_i}{\mu_{i+1}}\pi_i - \frac{\lambda_{i-1}}{\mu_{i+1}}\pi_{i-1} \quad \forall_{i>0}, \quad \text{and} \quad \pi_1 = \frac{\lambda_0}{\mu_1}\pi_0 \quad \overset{\text{by induction}}{\Rightarrow} \quad \pi_{i+1} = \pi_0 \prod_{j=0}^{i} \frac{\lambda_j}{\mu_{j+1}}. \tag{3.7}$$

If $\pi_0 = 0$ all states become 0, which is not possible. Consequently is a non zero probability π_0 a necessity. In words, the system must from time to time become idle. For the existence of the steady state must $\sum \pi_i$=1 be fulfilled, which is the case if and only if $\sum_{i=0}^{\infty} \prod_{j=0}^{i} \frac{\lambda_j}{\mu_{j+1}}$ is finite. This is the case if $\mu_i > 0 \; \forall_{i>0}$ and for some k all $\frac{\lambda_{k+j}}{\mu_{k+j+1}} < 1 \; \forall_j$ is true. In words, for stability no service rate μ_i may be zero (except μ_0), and the arrival rate λ_i needs to become and remain smaller than the service rate μ_{i+1} for some state k and all states above. Given stability we can calculate π_0

$$1 = \sum_{i=0}^{\infty} \pi_i = \pi_0 + \pi_0 \sum_{i=1}^{\infty} \prod_{j=0}^{i-1} \frac{\lambda_j}{\mu_{j+1}} \qquad \Rightarrow \qquad \pi_0 = \frac{1}{1 + \sum_{i=1}^{\infty} \prod_{j=0}^{i-1} \frac{\lambda_j}{\mu_{j+1}}} \tag{3.8}$$

and thereby all steady state probabilities π_i are defined via equation 3.7.

Rearranging 3.7 we get $\lambda_i p_i - \mu_{i+1} p_{i+1} = \lambda_{i-1} p_{i-1} - \mu_i p_i = \cdots = \lambda_0 p_0 - \mu_1 p_1 = 0$, the recursion of all the global steady state equilibrium equations $\lambda_{i-1} p_{i-1} = \mu_i p_i$, which result from splitting all states into any two closed sets of adjacent states. Here, for the one dimensional case, these are the boundary equations among any two adjacent states.

In *matrix form* the general birth-and-death process is defined by

$$Q = \begin{bmatrix} -\lambda_0 & \lambda_0 & & & \\ \mu_1 & -(\lambda_1+\mu_1) & \lambda_1 & & \\ & \mu_2 & -(\lambda_2+\mu_2) & \lambda_2 & \\ & & \ddots & \ddots & \ddots \end{bmatrix} \quad \overset{\vec{\pi}Q=\vec{0}}{\Rightarrow} \quad \pi_{i+1} = \pi_0 \prod_{j=0}^{i} \frac{\lambda_j}{\mu_{j+1}}. \tag{3.9}$$

All *M/M/x*-systems are special cases of the birth-and-death family. If the *M*s are generalised to *MAP*s we get the *quasi-birth-and-death* (QBD) family, where memoryless is restricted to renewal instances. Note also, that the birth-and-death as well as the QBD theory is not restricted to infinite systems. If at any state i the arrival rate becomes zero the states above cannot be reached and thus, also finite sets of connected Markov states define QBD processes.

3.1.1 *M/M/n* queueing systems

How an *M/M/1* system is solved has already been shown in section 1.4, where Markov chain models for queueing systems were introduced. The extension to n servers is straightforward and depicted in figure 3.2, which evidently yields the *M/M/1* system for n=1. The *M/M/n* queueing system is a

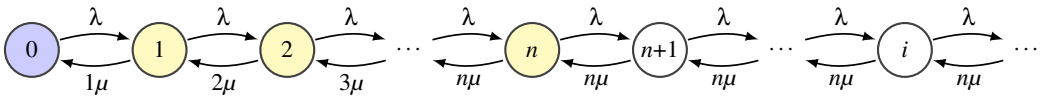


Figure 3.2: State transition diagram of an *M/M/n* queueing system

birth-and-death system where the rates λ_i=$\lambda \; \forall_i$ and μ_i=$n\mu \; \forall_{i \geq n}$ become state independent for $i \geq n$. Consequently is this system stable for any $\lambda < n\mu$. With multiple servers the sojourn time versus holding time divergence is also reflected by the different load definitions to be found in the literature: the offered load in Erlang, $\rho = \lambda h = \frac{\lambda}{\mu}$, versus the system load, $\varrho = \frac{\lambda}{n\mu} = \frac{\rho}{n}$.

The steady state equilibrium equations to be solved in order to get the state probabilities are

$$p(i-1)\lambda = p(i)\, i\mu \quad \Rightarrow \quad p(i) = \tfrac{\rho}{i} p(i-1) = \tfrac{\rho^i}{i!} p(0), \quad p(n) = \tfrac{\rho^n}{n!} p(0) \quad i \leq n \tag{3.10}$$

$$p(i-1)\lambda = p(i)\, n\mu \quad \Rightarrow \quad p(i) = \tfrac{\rho}{n} p(i-1) = \tfrac{\rho^i}{n!\, n^{(i-n)}} p(0) = \left(\tfrac{\rho}{n}\right)^j p(n) \quad i \geq n,\; j = i-n \tag{3.11}$$

if we use the boundaries between adjacent states to define the global equilibrium equations, and the offered load in Erlang $\rho = \frac{\lambda}{\mu}$ to state the load. Using $\sum p(i) = 1$ and $\sum_0^\infty a^i = \frac{1}{1-a}$ we get p_0.

$$1 = \sum_{i=0}^{n-1} \frac{\rho^i}{i!} p(0) + \sum_{i=n}^{\infty} \frac{\rho^i}{n!\,n^{(i-n)}} p(0) = p(0) \left(\sum_{i=0}^{n-1} \frac{\rho^i}{i!} + \frac{\rho^n}{n!} \sum_{j=0}^{\infty} \left(\frac{\rho}{n}\right)^j \right) = p(0) \left(\sum_{i=0}^{n-1} \frac{\rho^i}{i!} + \frac{\frac{\rho^n}{n!}}{1-\frac{\rho}{n}} \right)$$

$$\Rightarrow \quad p(0) = \frac{1}{\frac{\frac{\rho^n}{n!}}{1-\frac{\rho}{n}} + \sum_{i=0}^{n-1} \frac{\rho^i}{i!}} = \frac{1-\frac{\rho}{n}}{\frac{\rho^n}{n!} + (1-\frac{\rho}{n}) \sum_{i=0}^{n-1} \frac{\rho^i}{i!}} \overset{\rho \to n\varrho}{=} \frac{1}{\frac{(n\varrho)^n}{n!(1-\varrho)} + \sum_{i=0}^{n-1} \frac{(n\varrho)^i}{i!}} \tag{3.12}$$

Another important property is the probability for all servers being busy

$$p(i \geq n) = \sum_{i=n}^{\infty} \frac{\rho^i}{n!\,n^{(i-n)}} p(0) = \frac{p(n)}{1-\frac{\rho}{n}} = \frac{\frac{\rho^n}{n!}}{\frac{\rho^n}{n!} + (1-\frac{\rho}{n}) \sum_{i=0}^{n-1} \frac{\rho^i}{i!}} \overset{\rho \to n\varrho}{=} \frac{1}{1 + \frac{n!(1-\varrho)}{(n\varrho)^n} \sum_{i=0}^{n-1} \frac{(n\varrho)^i}{i!}} \tag{3.13}$$

known as the *Erlang_C formula*, expressing the probability that an arriving customer has to wait for being served. Note, this neither expresses how long an arriving customer will wait in average $\mathrm{E}[T_w]$, nor how many customers in average are waiting in the queue $\mathrm{E}[Q]$.

This *all servers busy* probability can be used to define a *macro state* $p(i<n) = 1 - p(i \geq n)$ that joins all states where at least one server is idle. Multiplied with the mean service duration $\tau_h = \frac{1}{\mu}$ this yields the flow time portion of the customers that do not need to wait for being served. With this macro state we can draw the state flow diagram as shown in figure 3.3. This simplified flow

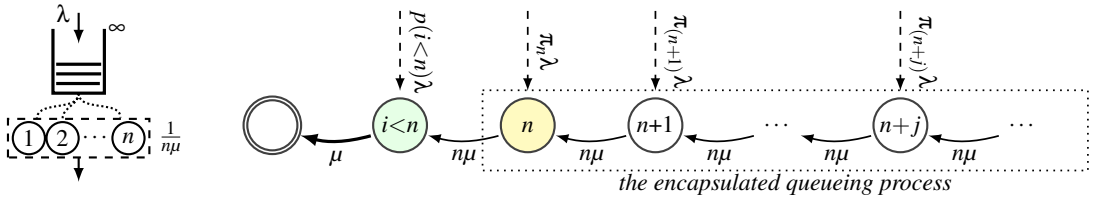


Figure 3.3: State flow diagram of the *M/M/n*-FIFO queueing system's queue

diagram equals that of an *M/M/1*-FIFO queueing system with service rate $n\mu$, except for the serving of the test customer itself into the absorbing state, which evidently happens with rate μ. Note that with multiple serves FIFO-queueing does not yield a FIFO system: due to varying service times the output order may differ from the arrival order. *First come first serve* (FCFS) better indicates such a system. However, the mean flow time $\mathrm{E}[T_f]$ results from

$$f \;=\; \mathrm{E}[T_f] \;=\; \mathrm{E}[T_w] \;+\; \mathrm{E}[T_h] \;=\; \underbrace{w}_{\text{waiting time}} \;+\; \underbrace{\tfrac{1}{\mu}}_{\text{serving time}} \tag{3.14}$$

and the enclosed mean waiting time w can be calculated via equation 3.11, using $\sum_1^\infty i\,a^i = \frac{a}{(1-a)^2}$,

$$w = \mathrm{E}[T_w] = \tfrac{1}{\lambda}\mathrm{E}[Q] = \tfrac{1}{\lambda} \sum_{j=1}^{\infty} j\,\pi_j = \tfrac{p(n)}{\lambda} \sum_{j=1}^{\infty} j\,\varrho^j = \tfrac{p(n)}{\lambda} \frac{\varrho}{(1-\varrho)^2} \overset{\varrho \to \frac{\rho}{n}}{=} \frac{p(n)}{n\mu(1-\frac{\rho}{n})^2} \tag{3.15}$$

or using the similarity with *M/M/1* via the conditional mean waiting time w^*, being the flow time $\mathrm{E}\left[T_f^*\right]$ of the *encapsulated queueing process* marked in figure 3.3

$$w^* = \mathrm{E}\left[T_f^*\right] = \tfrac{\mathrm{E}[X^*]}{\lambda} = \tfrac{1}{\lambda} \sum_{j=1}^{\infty} j\,\pi_j^* = \tfrac{\pi_0^*}{\lambda} \sum_{i=1}^{\infty} i\,\varrho^i = \tfrac{1-\varrho}{\lambda} \frac{\varrho}{(1-\varrho)^2} = \frac{1}{n\mu - \lambda} \tag{3.16}$$

independent of how the load was defined in the first place. Comparing the results we recognise that the required scaling factor

$$\frac{w}{w^*} = \frac{p(n)}{1-\frac{\rho}{n}} = p(i \geq n) \quad \Rightarrow \quad w = p(i \geq n)\, w^* = \frac{p(i \geq n)}{n\mu - \lambda} \tag{3.17}$$

is precisely the probability that an arriving client has to wait for service, $p(i \geq n)$. Consequently is the decomposing of the system in a serving and a queueing part possible, at least for the FIFO queueing discipline. Note that for the encapsulated queueing process there exists no server that takes a client from the queue, and consequently is $Q^* \equiv X^*$ and thus $w^* \equiv f^*$. This case appears whenever the departure from the queue occurs at service completion and not at service initiation.

Finally, we get the mean flow time $f = \mathrm{E}[T_f]$ by adding one mean holding time $h = \frac{1}{\mu}$ to the above achieved mean waiting time w in order to include the time required to serve the test load.

$$f \;=\; \mathrm{E}[T_f] \;=\; \frac{p(i \geq n)}{n\mu - \lambda} \;+\; \frac{1}{\mu} \;=\; \frac{1}{\lambda}\Bigg(\underbrace{\overbrace{\frac{\frac{\rho}{n}\, p(n)}{(1-\frac{\rho}{n})^2}}^{\mathrm{E}[Q]} \;+\; \rho}_{\mathrm{E}[X]} \Bigg)$$

This equivalently results if we apply Little's law on $\mathrm{E}[X]$ calculated directly via the equilibrium equations 3.10 and 3.11 as

$$\begin{aligned}
\mathrm{E}[X] = \sum_{i=1}^{\infty} i\, p(i) &= p(0)\Big[\sum_{i=1}^{n} i\, \tfrac{\rho^i}{i!} \;+\; \sum_{i=n+1}^{\infty} i\, (\tfrac{\rho}{n})^{(i-n)}\, \tfrac{\rho^n}{n!} \Big] \\
&= p(0)\Big[\rho \sum_{i=1}^{n} \tfrac{\rho^{(i-1)}}{(i-1)!} \;+\; \tfrac{\rho^n}{n!} \sum_{j=1}^{\infty} (n+j)\, (\tfrac{\rho}{n})^j \Big] \\
&= p(0)\Big[\rho \sum_{i=0}^{n-1} \tfrac{\rho^i}{i!} \;+\; \tfrac{\rho^n}{n!} \Big(n \sum_{j=1}^{\infty} (\tfrac{\rho}{n})^j + \sum_{j=1}^{\infty} j\, (\tfrac{\rho}{n})^j \Big) \Big] \\
&= p(0)\Big[\rho \sum_{i=0}^{n-1} \tfrac{\rho^i}{i!} \;+\; \tfrac{\rho^n}{n!} \Big(\tfrac{n\,\frac{\rho}{n}}{1-\frac{\rho}{n}} \;+\; \tfrac{\frac{\rho}{n}}{(1-\frac{\rho}{n})^2} \Big) \Big] \\
&= \underbrace{p(0)\tfrac{\rho^n}{n!}}_{p(n)} \tfrac{\frac{\rho}{n}}{(1-\frac{\rho}{n})^2} \;+\; p(0)\,\rho \Big(\underbrace{\tfrac{\frac{\rho^n}{n!}}{1-\frac{\rho}{n}} \;+\; \sum_{i=0}^{n-1} \tfrac{\rho^i}{i!}}_{=\frac{1}{p(0)} \text{ as shown in 3.12}} \Big) \\
&= \frac{\frac{\rho}{n}\, p(n)}{(1-\frac{\rho}{n})^2} \;+\; \rho \\
&= \mathrm{E}[Q] \;+\; \rho
\end{aligned} \tag{3.18}$$

where we again used $\sum_0^\infty a^i = \frac{1}{1-a}$, in particular $\sum_1^\infty a^i = \sum_0^\infty a^i \; - 1 = \frac{a}{1-a}$ and $\sum_1^\infty i\, a^i = \frac{a}{(1-a)^2}$, to solve the infinite sums in the third line.

The last equation in equation 3.18 results generally from Little's law: $x = \lambda f = \lambda w + \frac{\lambda}{\mu} = q + \rho$. Because Little's law is always applicable, this holds for any arrival and service time distribution. In other words, because the mean system filling $\mathrm{E}[X]$ has to equal the mean queue filling $\mathrm{E}[Q]$ plus the mean number of occupied servers, the latter must equal the load measured in Erlang. Because $\varrho = \frac{\rho}{n}$ is the average loading of the system, it is also the average utilization of the system if $\varrho \leq 1$, and also expresses the average utilization of each server if the server assignment is performed unbiased. Decomposing the system in a processing and a queueing part is not the most straightforward approach. Still, it is elegant and a common method to approximate complex systems exploiting the general applicability of Little's law.

Figure 3.4 shows how the system performance changes if the number of servers is increased while the entire system capacity $n\mu$ is kept constant. Latter is achieved by scaling down the service rate μ, dividing it by n, such that $\varrho=\frac{1}{n\mu}=\lambda$ becomes independent of n. Obviously, at low loads the

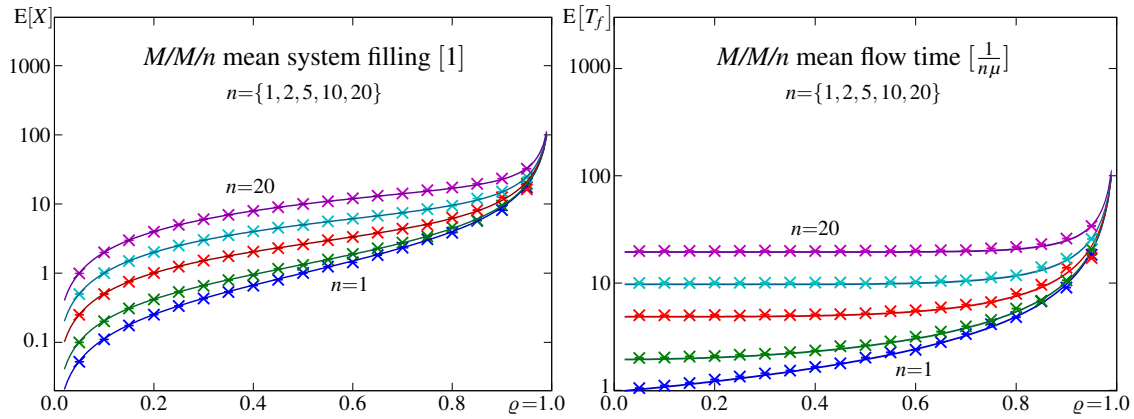


Figure 3.4: Mean system filling $\mathrm{E}[X]$ and flow time $\mathrm{E}[T_f]$ of $M/M/n$ first-come-first-serve (FCFS) queueing systems shown for increasing number of servers $n=\{1,2,5,10,20\}$ in combination with simulation results ($\times$) all normalised to equal system load $\varrho=\lambda$ by setting $\mu=\frac{1}{n}$

mean flow time $f=\mathrm{E}[T_f]$ shown on the right is dominated by the increased holding time, $\mu=\frac{1}{n}$, which results from decreasing the service rate for increased server numbers in order to keep the system capacity constant. At high loads they diverge less because the waiting time component $w=\mathrm{E}[T_w]$ becomes dominant. In case of ten or more servers the mean flow time is nearly constant up to 80% load. This increased holding time also causes the increased system filling $\mathrm{E}[X]$ shown on the left. However, this only results from the longer serving interval required per server and not from an increased mean waiting time. Vice versa, as figure 3.5 shows, the mean waiting time w actually decreases dramatically with increasing server numbers n. In a pragmatic view, and in an operator's

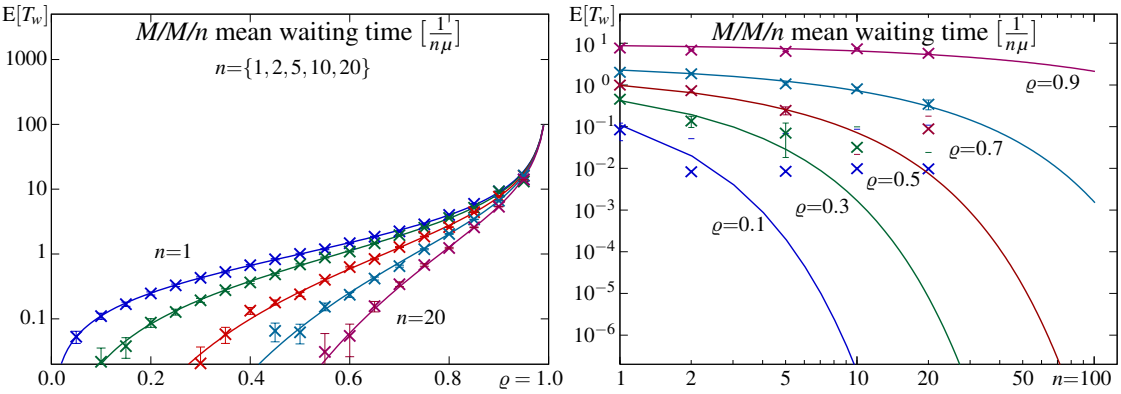


Figure 3.5: Mean waiting time $\mathrm{E}[T_w]$ of $M/M/n$ queueing systems shown for increasing number of servers $n=\{1,2,5,10,20\}$ over load and vice versa, for increasing load $\varrho=\{0.1,0.3,0.5,0.7,0.9\}$ over number of servers, again normalised to equal system load ϱ by setting $\mu=\frac{1}{n}$

perspective, the best service is provided for the least number of servers possible, being a single server system where the sole server offers a service rate equal to the joint service rate (capacity) of a multi-server system. Consequently, if the costs of using a faster server rise less than linear with its serving rate, the faster server is the more efficient choice. However, less servers cause a less homogeneous service in case of dynamic loads. If the mean waiting time w is more relevant than the holding time h, for example where long queues cause a bad reputation due to discouraged

customers or when high jitter has to be avoided because consistent timing is demanded, a system with an adequately chosen number of servers is the better choice.

In general, multi-server systems allow a trade off among mean flow time f and mean waiting time w. If power consumption is relevant, multi-server systems offer a simple or even implicit saving option: surplus server capacity can be switched off when the average load is low. In section 3.2 on finite queueing systems we will see that the *blocking probability*, a quality degradation commonly assumed more sever than jitter, also decreases for increased number of servers.

3.1.2 Queueing disciplines

The first come first serve (FCFS) policy results from *first in first out* (FIFO) queueing. Note that FCFS refers to the system, whereas FIFO refers to the queue within the system. In case of multiple servers and random service times, FIFO-queueing does not grant a FIFO-system because it does not assure that the loads depart in the order they arrived. FIFO is the most natural queueing discipline, and therefore it is the scheme by default assumed and discussed. In practice other schemes may occur as well. The most general queueing disciplines are sketched in figure 3.6. A priory, we note that *the*

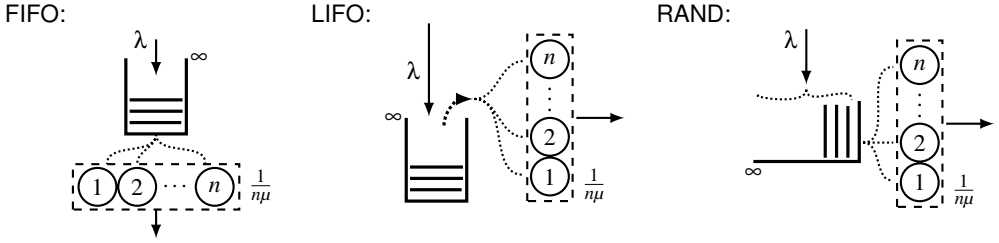


Figure 3.6: Different queueing disciplines: FIFO, LIFO, and random queueing

processing order, which is jointly determined by the queueing, scheduling, and serving/processing discipline, *cannot affect the average system performance of loss-less systems*. Only the higher moments of the waiting and flow time as well as the inter-departure process may depend on the departures ordering.

Concerning *processing disciplines* we broadly restrict the discussion to the *non-preemptive* discipline. Particularly because with multiple servers a *preemption policy* would be required to define which server, out of the n being busy, actually becomes preempted at an arrival instant. In case of non-preemptive disciplines only the currently queued customers are affected by later arrivals, and because the queueing discipline applies per queue, no additional information is required to define these systems. In addition, fractions of data-loads can commonly not be processed well due to the widely used transmission coding and error detection mechanisms. Thus, only the resource wasting and therefore congestion prone *pre-empt and re-start* mechanism would be a feasible pre-emption policy for data transmission systems. The *pre-empt and resume* policy is efficiently feasible only prior packeting and framing of data loads, which occurs in the transmit stack of traffic sources.

M/M/n/FIFO

The *first in first out* (FIFO) queueing state flow diagram shown in figure 3.3 reveals that for FIFO queueing the waiting process is a pure dying process, where only transitions toward the absorbing state exist. The number of negative exponentially distributed service phases contributing to the flow time component $T_f^*(j)$ is given by j, the number of clients waiting when the test load enters the queue plus one. Latter considering the residual service time till the next service completion, which is negative exponentially distributed with mean $\frac{1}{n\mu}$ because n identical memoryless service phases with mean μ each, equal one with mean $n\mu$ as long as no server becomes idle. Thus, the conditional flow time components $T_f^*(j)$ contributing to T_w are Erlang$^{[j+1]}(n\mu)$ distributed.

To get rid of the conditioning these flow time components need to be summed over all conditions, each contributing $f_{T_f^*(j)}(\tau)$ scaled by the according entry probability p_j^a. Poisson arrivals see time averages (PASTA), thus $p_j^a \equiv p(n+j)$ given by equation 3.11.

$$p_j^a = p(n+j) = p(n)\varrho^j \qquad\qquad f_{T_f^*(j)}(\tau) = \frac{n\mu(n\mu\tau)^j}{j!}e^{-n\mu\tau}$$

The waiting time density conditioned to $T_w > 0$ results as[1]

$$f_{T_w^*}(\tau) = \sum_{j=0}^{\infty} p_j^a\, f_{T_f^*(j)}(\tau) = \sum_{j=0}^{\infty} p(n)\,\varrho^j\,\frac{n\mu(n\mu\tau)^j}{j!}e^{-n\mu\tau} = p(n)\,n\mu\,e^{-n\mu\tau}\sum_{j=0}^{\infty}\frac{(\lambda\tau)^j}{j!}$$
$$= p(i\geq n)\,(n\mu-\lambda)\,e^{-(n\mu-\lambda)\tau} \tag{3.19}$$

where we use equation 3.13: $p(n) = (1-\varrho)\,p(i\geq n)$ and $\sum \frac{\alpha^j}{j!} = e^{\alpha}$. Entries to states not comprising a waiting time were implicitly considered because $\sum p_j$ does not sum to one, but to $p(i\geq n)$, being the probability that an arrival needs to wait as given by the Erlang_C formula. The thereby missing part constitutes a Dirac delta impulse at $t=0$ in the waiting time *pdf* and a step at $t=0$ in the *cdf*, both with magnitude $p(i<n) = 1 - p(i\geq n)$.

In figure 3.7 and figure 3.8 we show simulation results (histograms) to depict the *load dependent pdf* and *cdf* of the waiting time T_W, and recognise that the *pdf* is composed of a Dirac impulse

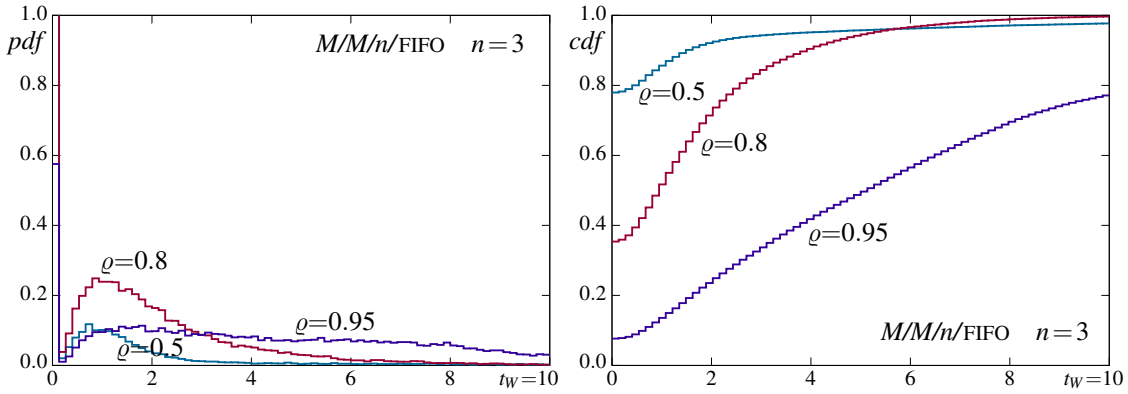


Figure 3.7: *M/M/n/*FIFO waiting time T_W *pdf* and *cdf* histograms for $n = 3,\ \varrho = [0.5, 0.8, 0.95]$.

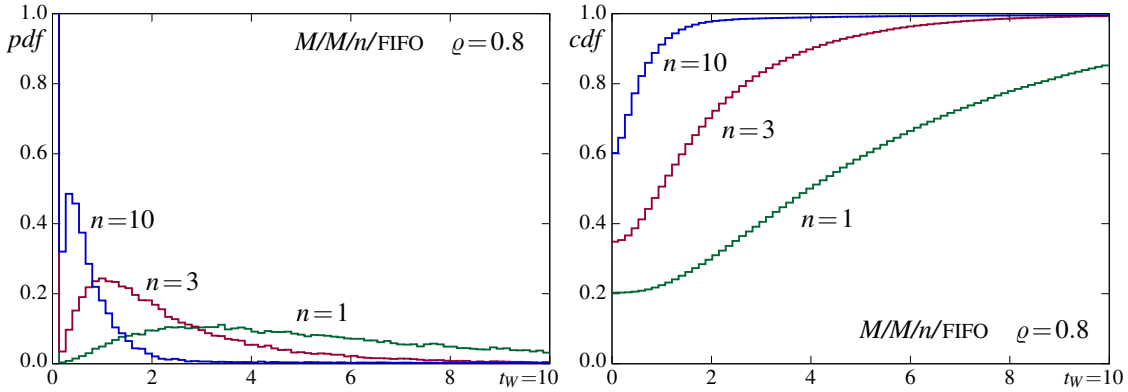


Figure 3.8: *M/M/n/*FIFO waiting time T_W *pdf* and *cdf* histograms for $\varrho = 0.8,\ n = [1, 3, 10]$.

at $t_W = 0$ with intensity $p(i<n)$ and a seemingly hypo-exponential component (zero at $t_W = 0$) that widens (changes its shape) with the system load. The *cdf* on the right shows that the waiting time

[1] the derivation is based on [34, page 410/411] on *M/M/1* flow time distribution, substituting $n \to j$, $\mu \to n\mu$

T_W is not negative exponentially distributed: there exists a non-zero probability for $\mathrm{P}[T_W=0] = p(i<n) = 1-p(i\geq n)$ that decreases with rising system load, approaching zero for $\varrho \to 1$. The *pdf* shown in figure 3.8 indicates that the conditional waiting time narrows for increasing server numbers. This is evident because for $n \to \infty$ we must get $T_W \to 0$. The *cdf* on the right shows that for the same system load ϱ the probability for no waiting $F_{T_W}(0)$ increases with the number of servers n, but also that even for $n=1$ the unconditioned waiting time T_W is not negative exponentially distributed.

M/M/n/LIFO

In case of *last in first out* (LIFO) queueing, which causes a *non-preemptive last come first serve* (np-LCFS) system, an arrival becomes immediately served if $X(t) < n$ at the arrival instant t, or it enters the queue at its head, pushing back all the already waiting loads by one place. Such a queue is commonly called a *stack* because it resembles a stack of documents where every new arriving document is put on the stack's top and for processing the documents are also taken from the top, one at a time, independent on how many documents have been added since the last document has been taken away for processing. The resultant flow diagram is shown in figure 3.9, and we note that here

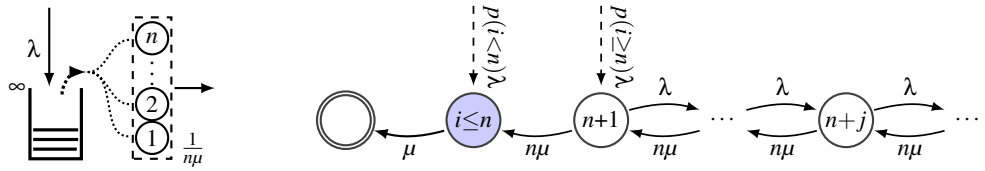


Figure 3.9: The *M/M/n*-LIFO state flow process

we consider the state of the queue after the arrival, in contrast to figure 3.3 where we considered the state of the queue prior arrival. With LIFO the number of customers in the queue at the arrival instance t is irrelevant, only those arriving while the test customer is waiting in the queue determine its waiting time, not those the test customer pushed back.

The common approach to solve this is to calculate the mean number of arrivals in between arriving to the queue and being served times the mean inter-departure time t_d. Evidently, $X(t)>n$ is given for the entire waiting epoch and thus, $t_d(i \geq n) = \frac{1}{n\mu}$ as long as a load is waiting in the queue. Consequently is the mean number of arrivals in between adjacent departures $a_d(i \geq n)$ equal the arrival rate λ times t_d, and this actually defines the system load ϱ.

$$a_d(i \geq n) = \mathrm{E}[A(T_D, i \geq n)] = \frac{\lambda}{n\mu} = \varrho$$

With LIFO queueing every such arrival needs to be served prior the test load, and thus each adds another inter-departure interval during which more loads may arrive. This circumstance yields the mean number of arrivals a_w that occur during the waiting time of a test load.

$$a_W = \sum_{i=0}^{\infty} a_d(i \geq n)^i = \frac{1}{1-\varrho} \tag{3.20}$$

Multiplied with the applicable mean inter-departure time $t_d(i \geq n) = \frac{1}{n\mu}$ we get the conditional mean waiting time $\mathrm{E}[T_W(i \geq n)]$ for test loads that need to wait. Further multiplied with the probability that loads arrive at times when they need to wait, given by $p(i \geq n)$, we finally get the unconditional mean waiting time $\mathrm{E}[T_W]$.

$$\mathrm{E}[T_W(i \geq n)] = \frac{1}{n\mu - \lambda} \quad \Rightarrow \quad \mathrm{E}[T_W] = \frac{p(i \geq n)}{n\mu(1-\varrho)}$$

The result equals equation 3.17 derived for FIFO queuing. This *analytic identity* indicates that *the mean waiting time* $\mathrm{E}[T_W]$ *does not depend on the queueing discipline*. And likewise, also the state probabilities p_i will not depend on the queueing discipline, only the positioning of different clients within the queue is affected.

To get the distribution of the waiting time T_W we again start with the conditional $T_W^*(i)>0$. The arrival of the client being the first to become queued also starts the *busy period*, if we define it as the period during which arrivals become queued or the time span while all servers are persistently busy, correspondingly. Due to LIFO queueing the busy period equals the waiting time T_W^* of that client, and because clients in the queue at the arrival instant have no influence on the waiting time of the arriving client, this needs to apply for all clients that become queued. Thus, the conditional waiting time distribution equals the distribution of the busy period, $F_{T_W^*} \equiv F_{T_B}$. Next we recognise that the busy period T_B of any *M/M/n* system is the same, and that it equals that of *M/M/1* if we set $\mu^{[M/M/1]} = \sum \mu_k^{[M/M/n]} = n\mu$, if $\mu_k{=}\mu\,\forall_k$, because during a busy period no server-idle intervals interfere with the negative exponentially distributed holding times at every server.

This can be utilized to get the waiting time distribution $F_{T_W}(\tau)$ via the *Kendall's functional equation*, applicable for any *M/G/1* system. Here, inserting the Laplace transform of the negative exponentially distributed *virtual* holding times $T_h(s) = \frac{n\mu}{s+n\mu}$ provided by the n servers in parallel.

$$T_B(s) = T_h\big(s+\lambda-\lambda T_B(s)\big) \quad \longrightarrow \quad T_B(s) = \tfrac{1}{2\lambda}\big(s+\lambda+n\mu-\sqrt{(s+\lambda+n\mu)^2-4\lambda n\mu}\big)$$

However, there exist an alternative combinatorial path toward the solution not based on transforms, which is briefly sketched here.[2]

Let j state the random number of arrivals pushing back the test client while it is waiting. Than the number of inter-departure times until the test client becomes served is $j{+}1$, each equalling a service time of the *M/M/1* system with mean service rate $n\mu$ because no idle-server intervals exist during a busy period and all holding times are negative exponentially distributed. The probability for the busy period T_B to be within $[\tau\,..\,\tau+\Delta\tau]$ can be expressed probabilistically as

$$\mathrm{P}[T_B \in (\tau,\tau+\Delta\tau)] = \sum_{j=0}^{\infty} \mathrm{P}\big[j \text{ arrivals } \cap\ (j{+}1) \text{ departures in } [0\,..\,\tau{+}\Delta\tau] \,\big|\, i_a(t){>}i_d(t)\,\forall_{t<\tau}\big]$$

where $i_a(t){>}i_d(t)\,\forall_{t<\tau}$ expresses the condition that the *M/M/1* system may never becomes empty during the busy period, requiring more arrival events $i_a(t)$ than departure events $i_d(t)$ at any time t prior τ has passed.

During the busy period occur state changing events negative exponentially distributed with mean rate $\lambda{+}n\mu$, and we have $2j{+}1$ events in total, but only $2j$ inter-event intervals. Among these events, arrivals and departures occur with probabilities $\frac{\lambda}{\lambda+n\mu}$ and $\frac{n\mu}{\lambda+n\mu}$. For $\Delta\tau \to 0$ we get $\mathrm{P}[T_B \in (\tau,\tau+\Delta\tau)] \to$ busy period *pdf* $f_{T_B}(\tau)$ as

$$f_{T_B}(\tau) = \sum_{j=0}^{\infty} \underbrace{\frac{\tau^{2j}(\lambda+n\mu)^{2j+1}}{(2j)!}e^{-(\lambda+n\mu)\tau}}_{\substack{\text{probability of } 2j{+}1 \text{ events}\\ \text{at rate } \lambda{+}n\mu \text{ in } \tau \text{ time units,}\\ \text{including both boundary}\\ \text{events}}} \underbrace{\left(\frac{\lambda}{\lambda+n\mu}\right)^{j}\left(\frac{n\mu}{\lambda+n\mu}\right)^{j+1}}_{\substack{\text{probability of } j \text{ arrivals}\\ \text{and } j{+}1 \text{ departures}\\ \text{among the } 2j{+}1 \text{ events}}} \underbrace{\left(\binom{2j}{j}-\binom{2j}{j+1}\right)}_{\substack{\text{number of random walks with}\\ 2j \text{ steps not reaching an idle}\\ \text{server prior the } (2j{+}1)^{\text{th}} \text{ step}\\ \text{(non-conforming subtracted)}}}$$

$$= e^{-(\lambda+n\mu)\tau}\sum_{j=0}^{\infty}\lambda^j(n\mu)^{j+1}\tau^{2j}\left(\frac{1}{j!\,j!}-\frac{1}{(j-1)!\,(j+1)!}\right) = e^{-(\lambda+n\mu)\tau}n\mu\sum_{j=0}^{\infty}\frac{(\sqrt{\lambda n\mu}\,\tau)^{2j}}{j!(j+1)!}$$

[2] Thanks to the lucid explanations of my co-advisor Prof. Karl Grill, compare [65, p.39] referring to N.U. Prabhu, 1960.

where we use the *mirroring theorem* for random walks to get rid of the non-conforming walks. Using the *first modified Bessel function* $I_1(x) = \sum_{k=0}^{\infty} \frac{1}{k!\,(k+1)!} \left(\frac{x}{2}\right)^{2k+1}$ we can replace the infinite sum.

$$f_{T_B}(\tau) = e^{-(\lambda+n\mu)\,\tau}\, \frac{n\mu}{\sqrt{\lambda n\mu}\,\tau}\, I_1(2\sqrt{\lambda n\mu}\,\tau) = \frac{I_1(2\sqrt{\lambda n\mu}\,\tau)}{\sqrt{\varrho}\,\tau}\, e^{-(\lambda+n\mu)\,\tau} \tag{3.21}$$

The unconstrained waiting time distribution $f_{T_W}(\tau)$ results from combining the above result with a Dirac impulse at $\tau = 0$ with magnitude $p(i<n)$ to include the zero waiting contributed by arrivals served immediately.

$$f_{T_W}(\tau) = p(i<n)\delta(0) + p(i\geq n) f_{T_B}(\tau)$$

M/M/n/RAND

Finally, we sketch *random queueing* (RAND). Customers that arrive and need to be queued enter the queue at any place. Two policies are likely: (a) to choose any place in the entire queue, and serve the queue in a round robin fashion, skipping idle places, or (b) to choose the place depending on how many loads are currently in the queue. With infinite space only the latter is applicable. In that case all entry probabilities depend on the current state via the entry probability $e_j = p(i\geq n)\frac{j}{\mathrm{E}[Q]}\pi_{(i-1)}$. Figure 3.10 sketches the system and shows the state flow diagram for this *random queueing* system.

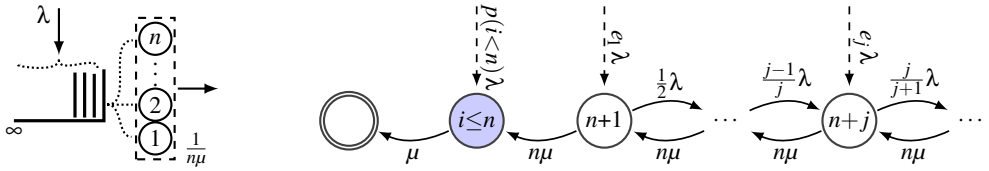


Figure 3.10: The *M/M/n*-RAND state flow process – random queueing

Again, we enumerate the states by the number after the arrival of the test load because it may be pushed back. Here both, loads already queued and those arriving while a test load is waiting, influence the waiting time.

Noting that unbiased random picking eliminates any initial queuing order, the above discussed random queueing equals *random serving* where a waiting load is picked randomly from the queue to enter service. This is sketched in figure 3.11, depicting the *M/M/n*-RAND state flow diagram for the *random serving* approach. The state flow diagram for random serving shows no splitting for the

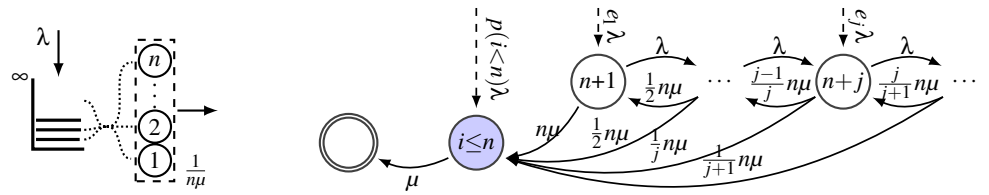


Figure 3.11: Alternative *M/M/n*-RAND state flow process – random serving

up-transitions because the queueing order is irrelevant. Without loss of generality, we may assume that arrivals push back the test load. This assumption allows us to use as entry probabilities $e_j = \pi_{(i-n)}$ because the test load can be assumed to define the end of the queue, and it also allows us to split the downward transitions by the selection probability $s(j) = \frac{1}{j}$ for the test load, and $s(<j) = \frac{j-1}{j}$ for some other load being served, causing the queue to shrink by one place.

The mean waiting time $\mathrm{E}[T_w]$ is evidently again not affected, only the higher moments change. Intuitively, random queueing is somewhere midway in between FIFO and LIFO, and the derivation

of the waiting time distribution similar for both cases, random queueing and random serving. Again, the waiting time distribution is composed of a Dirac impulse for no waiting plus weighted Erlang distributions that cover the different numbers of loads served while a test load is waiting. The formal approach is skipped, we directly proceed to simulated results.

In figure 3.12 the *pdf*s and *cdf*s of the waiting time T_W for *M/M/3/*FIFO, *M/M/3/*LIFO, and *M/M/3/*RAND are depicted by histograms achieved from simulation at system load $\varrho = 0.8$. As

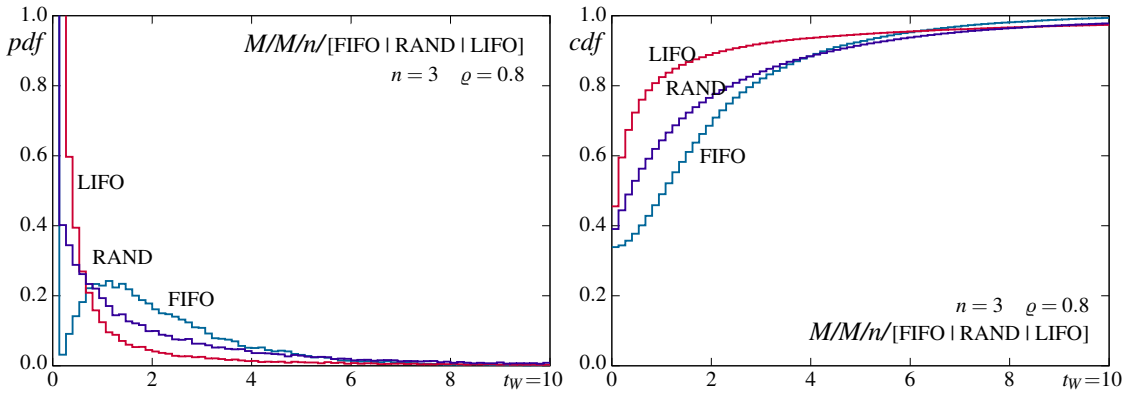


Figure 3.12: *M/M/n/*[FIFO|LIFO|RAND] waiting time T_W *pdf* and *cdf* for $n = 3,\ \varrho = 0.8$.

expected, the *M/M/3/*RAND queueing system shows a waiting time distribution that is somehow in between that of *M/M/3/*FIFO and *M/M/3/*LIFO depicted in figure 3.12. However, because for LIFO queueing short waiting times are far more likely than for FIFO queueing, its *pdf* does not show the ditch in between the Dirac impulse and some hypo-exponential component, and for random queueing the ditch reduces to an unsteadiness.

The distribution of the waiting time T_W depends on the number of servers n and the current system load ϱ, as already shown for *M/M/n/*FIFO in figure 3.7 and figure 3.8. Figure 3.13 now shows by simulation results that the *mean waiting time* $\mathrm{E}[T_W]$ actually does not depend on the queueing discipline (left side), and how the waiting time variation coefficient c_{T_W} changes over the system load (right side). That $\mathrm{E}[T_W]$ is independent of the queueing scheme has been argued and shown

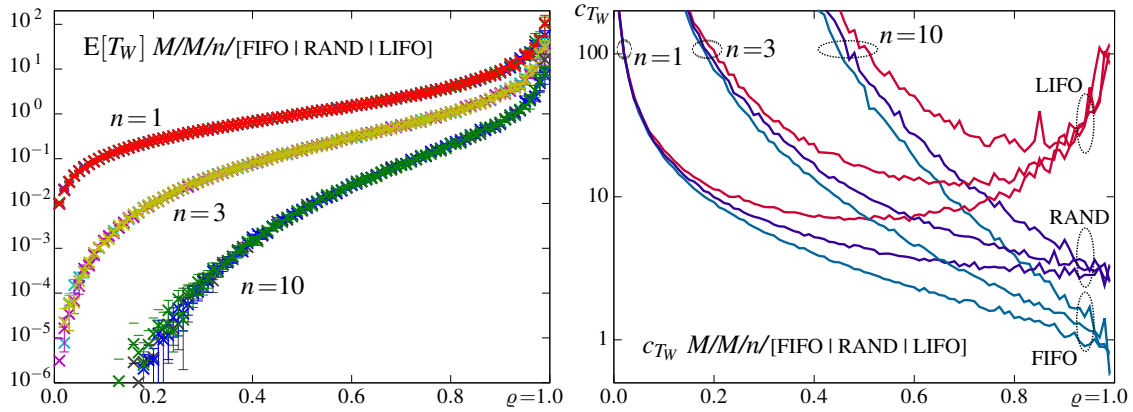


Figure 3.13: *M/M/n/*[FIFO|LIFO|RAND] $\mathrm{E}[T_W]$ and variation coefficient c_{T_W} over load.

analytically for FIFO and LIFO. Each of the visible traces for $n = [1, 3, 10]$ is composed of simulation results we got for FIFO, LIFO, and RAND queueing. That they are not separable proves that these queueing disciplines do not affect the mean waiting time. The right sub-figure in 3.13 is more interesting. First, the curves represent simulation results, here the calculated coefficient of variation of the waiting time c_{T_W}, and this is actually calculated from the same samples used on the left side to calculate $\mathrm{E}[T_W]$. Second, for clarity we do not show $\times$-es here, and thus, every bent represents a

sample result, the straight lines in between are irrelevant and are shown for the better appeal only. We recognise that the coefficient of variation depends on the number of servers n and the current system load ϱ, as expected.

At low loads the waiting time distribution is dominated by the idle times and rather independent of the queueing discipline. Vice versa, at high loads the waiting time distribution depends mostly on the queueing discipline and the number of servers becomes less relevant. Clearly visible, the waiting time variance for LIFO queueing approaches infinity for $\varrho \to 1$. This is the case because the duration of the busy period approaches infinity and thus the waiting time for loads that happen to arrive at the begin of the busy period becomes much longer than that of loads that arrive near the end of a busy period. In contrast, the coefficient of variation for FIFO queueing approaches one for $\varrho \to 1$. This is also clear because at $\varrho = 1$ the idle time would be zero and thus the waiting time composed of negative exponentially distributed service times only. As before exemplarily shown for $\varrho = 0.8$, the waiting time variance for RAND queueing is always in between that of the extremes, FIFO and LIFO. Based on the shown simulation results we can expect $c_{T_W}(RAND) \to const$ for $\varrho \to 1$, where *const* appears to be again independent of the number of servers. At system loads $\varrho < 1$ we see that the waiting time's coefficient of variation increases quite heavily with the number of servers n.

M/D/n/FIFO waiting time distribution

In case the service times are identical (deterministic) for all clients arriving to a multi-server system with FIFO queuing, the distribution of the waiting time can be derived. This system was readily introduced by A.K. Erlang [66, 1909] and formally solved by F. Pollaczek and C.D. Crommelin based on transforms and complex theory, about 15 years later [67].

The expression for the waiting time distribution of the *M/D/n/*FIFO system presented by A.K. Erlang

$$F_{T_w}(\tau) = \int_0^\infty F_{T_w}(t+\tau-\tfrac{1}{\mu}) \, \frac{\lambda^n t^{n-1}}{(n-1)!} e^{-\lambda t} \, \mathrm{d}t \tag{3.22}$$

is obtained by comparing the waiting times of every n^{th} customer, which arrive $\frac{\lambda^n t^{n-1}}{(n-1)!} e^{-\lambda t} \, \mathrm{d}x$ time units apart of each other. If such an arrival needs to wait, meaning all servers are busy when it arrives, it is served exactly $\frac{n}{\mu}$ time units after its n^{th} predecessor because in this time span the n servers complete exactly n clients, due to the constant holding time $T_h = \frac{1}{\mu}$. Thus, for any sequence of n^{th} arrivals the system performs alike a single server system with Erlang$^{[n]}$ distributed arrivals, i.e., an *E_n/D/1* system.

However, equation 3.22 is hard to solve for $n > 1$. The solutions presented by F. Pollaczek and C.D. Crommelin cause numeric difficulties (rounding errors) for large n and close to $\varrho = 1$, though approximations presented by A.K. Erlang can be used very well in these cases [68]. More recently a purely probabilistic approach has been presented by Franx [69, 2001], which promises less numeric difficulties.

$$\mathrm{P}[T_w < \tau] = e^{-\lambda(\frac{k}{\mu}-\tau)} \sum_{j=0}^{kn-1} Q_{kn-j-1} \frac{\lambda^j (\frac{k}{\mu}-\tau)^j}{j!} \qquad \text{for} \quad (k-1) < \mu\tau \le k \tag{3.23}$$

The parameter Q_m therein is the cumulative queue filling probability, $Q_m = \mathrm{P}[Q \le m] = \sum_{j=0}^{m} q_j$. The queue filling probabilities q_i can be calculated by solving the linear equation system given by

$$q_0 = \sum_{j=0}^{n} q_j \sum_{m=0}^{n-j} \frac{\rho^m}{m!} e^{-\rho} \quad \text{and} \quad q_i = \sum_{j=0}^{i+n} q_j \frac{\rho^{i+n-j}}{(i+n-j)!} e^{-\rho} \qquad \text{for} \quad i > 0 \tag{3.24}$$

where $\rho = \frac{\lambda}{\mu}$ is the load in Erlang (arrival rate $\times$ holding time), in contrast to the system load $\varrho = \frac{\lambda}{n\mu} = \frac{\rho}{n}$. Whether solving this infinite system of equations, $i \in \mathbb{N}_0$, is less bothersome than the

complexity and numeric issues of Crommeline's approach [68, 70] may be queried. If only $\mathrm{P}[T_w{=}0]$ and $\mathrm{E}[T_w]$ are required, the equations presented by F. Pollaczek can be used [68].

$$\mathrm{P}[T_w{=}0] = exp\Big\{ -\sum_{i=1}^{\infty} \frac{1}{i} e^{-i\rho} \sum_{j=ni}^{\infty} \frac{(i\rho)^j}{j!} \Big\} \tag{3.25}$$

$$\mathrm{E}[T_w] = \sum_{i=1}^{\infty} e^{-i\rho} \Big(\sum_{j=ni}^{\infty} \frac{(i\rho)^j}{j!} - \tfrac{n}{\rho} \sum_{j=ni+1}^{\infty} \frac{(i\rho)^j}{j!} \Big) \tag{3.26}$$

The *M/D/n/*FIFO example reveals the difficulties that multi server systems cause. Henceforth we consider systems with a single server for the general analysis. The impact of more servers may be assumed to be in principle similar to what we have shown above for the *M/M/n* queueing system.

3.1.3 *M/G/1* queueing systems

Assuming a non Markovian service time distribution we need to consider the time a client is already in service because the residual service time S_r is in general not independent of the time the client already has been served. To manage this we restrict the observation of the system to renewal instants t_n, where n indicates the chronological instant number (index). With *M/G/1* such renewals occur whenever the residual service time equals the entire service time, thus at the instants where serving starts. For these times instants we can draw the state transition diagram shown in figure 3.14. Actually, it is show for illustration and to present the duality with *GI/M/1* presented in section 3.1.4,

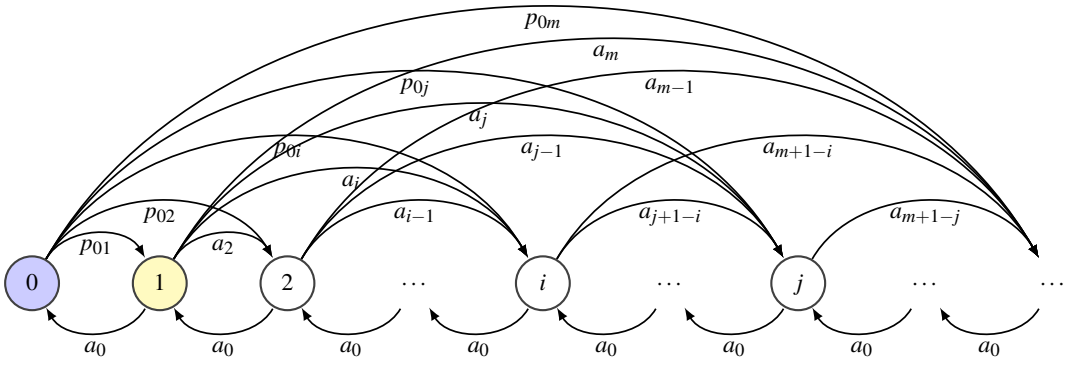


Figure 3.14: *M/G/1* state transition diagram

where the diagram is explained in more detail. For the here presented scheme to derive the system properties based on the *probability generating function* $P(z)$, it is not essentially required. However, a similar approach to that presented with *GI/M/1* would as well be applicable, though it requires a simple analysis of the idle process to cover the time the system remains idle in order to calculate the p_{0i} transitions. This idle process is not implicitly reflected in figure 3.14.

If the system is idle ($N_n{=}0$) an renewal instant occurs with the arrival of a client that instantly enters service. If the queue is not idle ($N_n{>}1$) these instants occur with client departures, when serving of the next client from the queue starts. During a service time any number of clients may arrive. Their random number is given by the distribution $A_n(\tau_S(n))$, which we will derive shortly. If at the n^{th} service begin $N_n{>}1$ clients were waiting these remain in the system except one, which is the one being served that leaves the system prior the next renewal instant. Thus, the number of clients at the next renewal instant $n{+}1$ result as

$$N_{n+1} = (N_n - 1)_+ + A_n \tag{3.27}$$

where $(\,..\,)_+ = max\{0, ..\}$ indicates that the term is non-negative. The probability that $N_n{=}k$ clients are in the system is $p_k = \mathrm{P}[N_n{=}k]$, which for the stationary system is independent of n. The generating function for the state probabilities is $P(z) = \sum p_k z^k$, and for $(N_n{-}1)_+$ we get

$$P_{(N_n-1)_+}(z) = p_0 + \frac{P(z)-p_0}{z} = \frac{P(z)-p_0(1-z)}{z}. \tag{3.28}$$

The generating function for the arrivals during a service interval $G_A(z)$ is the Laplace transform of the service time distribution $S(s)$ evaluated at $s = \lambda(1-z)$. From $P(z) = P_{(N_n-1)_+}(z)\,G_A(z)$ we get

$$\begin{aligned}
P(z) &= \frac{P(z)-p_0(1-z)}{z}\,S\big(s{=}\lambda(1{-}z)\big)\\
P(z)\,\big[\,S\big(s{=}\lambda(1{-}z)\big)-z\,\big] &= p_0(1-z)\,S\big(s{=}\lambda(1{-}z)\big)\\
P(z) &= \frac{p_0(1-z)\,S\big(s{=}\lambda(1{-}z)\big)}{S\big(s{=}\lambda(1{-}z)\big)-z}.
\end{aligned}$$

Inserting $p_0 = 1-\rho$, which results from $G_A(1) = 1$, we get the *Pollaczek-Kintchin formula*.

$$P(z) = \frac{(1-\rho)(1-z)\,S\big(s{=}\lambda(1{-}z)\big)}{S\big(s{=}\lambda(1{-}z)\big)-z} \tag{3.29}$$

Due to serving in the order clients arrive, the average number of clients that are in the system when any client n departs (N_{n+1}) must equal the average number of clients that arrive in the time that a client n spent in the system, being the flow time $\tau_F(n)$. Consequently, we get the Laplace transform of the flow time distribution from $T_F(s) = P\big(z{=}\frac{\lambda-s}{\lambda}\big)$ as

$$T_F(s) = \frac{(1-\rho)\,s\,S(s)}{s+\lambda\,(S(s)-1)}, \tag{3.30}$$

which is also a Pollaczek-Kintchin formula. And finally, the Laplace transform of the waiting time distribution $T_W(s)$ results from $T_F(s) = T_W(s)\,S(s)$ as

$$T_W(s) = \frac{(1-\rho)\,s}{s+\lambda\,(S(s)-1)}. \tag{3.31}$$

The means of these distributions are given by the so called *Pollaczek-Kintchin mean value formulas:*[3]

$$\mathrm{E}[N] = \rho + \frac{\lambda^2\,\mathrm{E}\big[S^2\big]}{2(1-\rho)} \tag{3.32}$$

$$\mathrm{E}[T_F] = \frac{\mathrm{E}[N]}{\lambda} = \frac{1}{\mu} + \frac{\lambda\,\mathrm{E}\big[S^2\big]}{2(1-\rho)} \tag{3.33}$$

$$\mathrm{E}[T_W] = \mathrm{E}[T_F] - \frac{1}{\mu} = \frac{\lambda\,\mathrm{E}\big[S^2\big]}{2(1-\rho)} \tag{3.34}$$

We note that these performance relevant metrics depend on the system load $\rho{=}\frac{\lambda}{\mu}{=}\frac{\mathrm{E}[S]}{\mathrm{E}[A]}$, which comprises the first moments of the arrival and service distribution, and the second raw moment $\mathrm{E}[S^2]{=}\,Var[S]{+}\,\mathrm{E}[S]^2$ of the service time distribution. Higher moments have no influence on these mean value metrics and thus any service time distribution with the same first two moments yields the same results and thus can be used interchangeably to model the service process if above metrics comprise all what is required. Finally, the variance of the waiting time σ_W^2 is defined by the mean system load ϱ and the second and third raw moments of the service process [65, page 88].

$$\sigma_W^2 = Var(T_W) = \frac{\lambda\,\mathrm{E}\big[S^3\big]}{3(1-\rho)} + \frac{\lambda^2\,\mathrm{E}\big[S^2\big]^2}{4(1-\rho)^2} \tag{3.35}$$

[3] A derivation from scratch is for example shown by W. J. Stewart in [34].

For *deterministic service times* with rate μ and deviation $\sigma_D{=}0$ we have $\mathrm{E}[S^2]{=}(\frac{1}{\mu})^2$ and get

$$\mathrm{E}[N_{M/D/1}] = \rho + \frac{\rho^2}{2(1-\rho)} = \frac{\rho}{1-\rho}\left(1-\frac{\rho}{2}\right). \tag{3.36}$$

Comparing this with $\mathrm{E}[N_{M/M/1}]{=}\frac{\rho}{1-\rho}$ we recognise that for the smoothest possible service time distribution the system occupation is decreased by the factor $(1{-}\frac{\rho}{2})$, with increasing improvement from none at $\rho{=}0$ to halve the occupation at $\rho{=}1$. Accordingly is also the flow and waiting time improved. In the next subsection we will see that *'the smoother the better'* evidently applies for the inter-arrival time distribution as well. For a *D/D/1* system we get $\mathrm{E}[N_{D/D/1}] < 1\ \forall_{\rho<1}$. As the mean occupation is strictly reduced, we can expect that in case of finite systems also the blocking probability is less with smoother processes.

Figure 3.15 shows the mean system filling and flow time of $M/G/1$ queueing systems with increasing service time variation. Curves for a coefficient of variation from $c_x{=}0$, being deterministic serving, up to $c_x{=}10$, being quite unsteady serving with variation $Var(S){=}c_x^2{=}100$, if we set $\mu{=}1$. The Markov case ($M/M/1$) is represented by $c_x{=}1$. The included simulation results basically approve

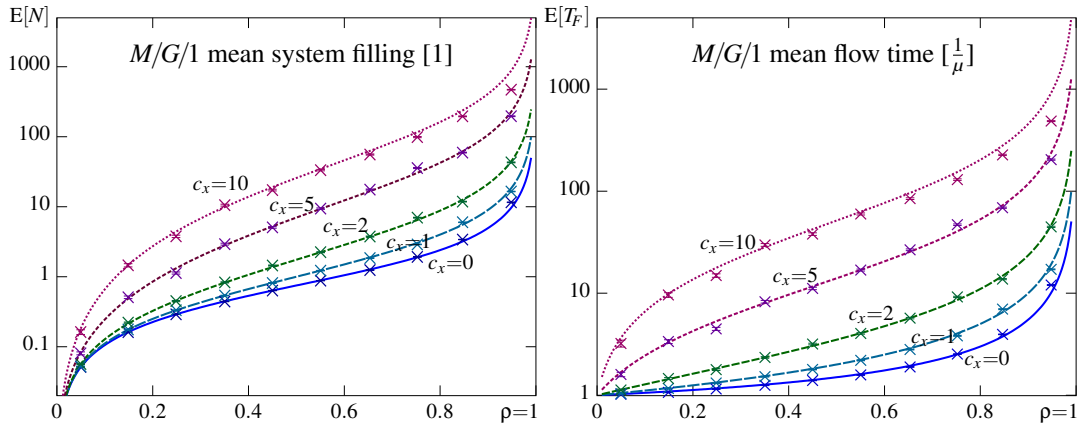


Figure 3.15: Mean system filling and flow time for and $M/G/1$ queueing system with coefficient of service time variance in $c_x = [0,1,2,5,10]$.

the analytic curves. Only for high loads and a coefficient of variation $c_x{>}2$ the 50.000 samples used to derive simulation results are insufficient to achieve a close match. The simulation routine used is presented and discussed in section 3.4.

We note that the difference between negative exponentially distributed serving (Markovian) and deterministic serving is rather small compared to the performance degradation that results from the service time variances we get for $c_x{>}2$. In particular, the more than exponential increase of the flow time at low loads expresses the fatal impact of highly varying service rates. No linear increase of the mean service rate can compensate this steep rise. Thus, we may conclude that: (a) smooth service rates are essential to achieve good performance at reasonable loads (system utilization), and (b) mean loads above 90% cause extreme queue filling and thereby huge flow times, which rarely are beneficial.

3.1.4 *GI/M/1* queueing systems

Assuming non Markovian but independent and identically distributed inter-arrival times we need to consider that the residual inter-arrival time is not independent of the time that passed since the last arrival. Again, we observe the system at renewal instants, which for $GI/M/1$ occur at arrival instants t_n, when the residual inter-arrival time equals the entire inter-arrival time.

In between successive arrivals, clients depart distributed with $S_n(\tau_A(n))$, depending on the current inter-arrival time $\tau_A(n)$. However, never can more depart than the number of clients N_{n-1} that were present at the last renewal instant. The number of clients in the system at the next renewal instant at t_n, just prior the client n arrives, results as

$$N_n = (N_{n-1} + 1 - S_n)_+ \tag{3.37}$$

where $(\,..\,)_+ = max\{0, ..\}$ again indicates that the term cannot be negative. The state transition diagram depicting this is shown in figure 3.16. The transition probabilities $p_{ij}\,\forall_{j>0}$ are given by

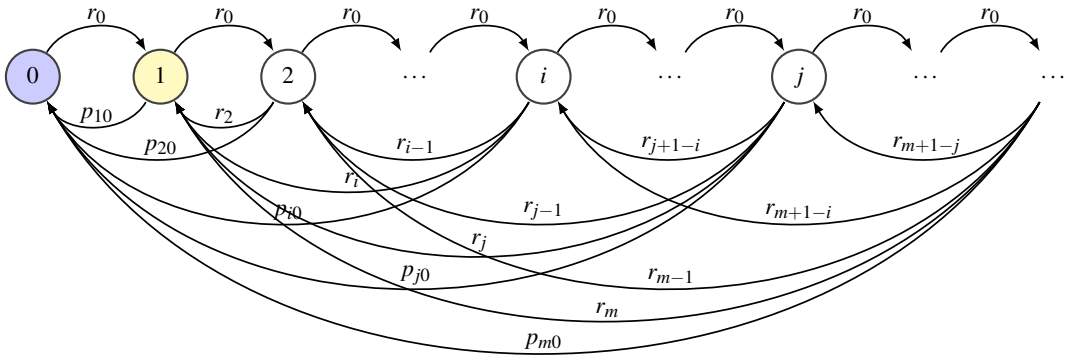


Figure 3.16: *GI/M/1* state transition diagram

the according departure rates r_k expressing the probability that in between two successive arrivals exactly k clients depart. If no departure occurs, which happens with rate r_0, the state increases by one due to the arrival that defines the observation instant. Never can the state increase by more within a single observation interval. In case exactly one departure occurs, which happens with rate r_1, the system stays in the same state because the single departure is compensated by the arrival. The remaining transition probabilities $p_{m0} = 1 - \sum_{i=0}^{m} r_i$ apply when all present clients become served and the system remains idle for some unknown duration until the next arrival occurs. This includes remaining in state 0 with the probability $p_{00} = 1 - r_0$.

The required rates r_k for k departures in between subsequent arrival instants, given that the queueing system does not become idle, can be calculated using

$$r_n = \int_{t=0}^{\infty} \frac{(\mu t)^n}{n!} e^{-\mu t} f_A(t)\,\mathrm{d}t \tag{3.38}$$

where $f_A(t)$ is the *pdf* of the general arrival process.

Note that the arrivals are no more Markovian and thus, the PASTA criterion (*Poisson arrivals see time averages*) no more applies. Consequently are the state probabilities $\pi_k = \mathrm{P}[N_n{=}k|t{=}t_n]$ at arrival instants not equal the global state probabilities $p_k = \mathrm{P}[N_n{=}k]$, even though in the steady state both are independent of n. If and only if the arrivals are Markovian is $\pi_k = p_k$. However, assuming $\rho < 1$, using the π_k states and the transition probabilities p_{ij} shown in figure 3.16 we can state the limiting state distribution by

$$\pi_0 = \sum_{i=0}^{\infty} \pi_i p_{i0} = \sum_{i=0}^{\infty} \pi_i \Big(1 - \sum_{j=0}^{i} r_j\Big) = 1 - \sum_{i=0}^{\infty} \sum_{j=0}^{i} \pi_i\, r_j$$

$$\pi_n = \sum_{i=0}^{\infty} \pi_{n-1+i}\, r_i$$

where we inserted $p_{i0} = 1 - \sum_{j=0}^{i} r_j$ and considered $\sum_{i=0}^{\infty} \pi_i = 1$ to get the expression for π_0. The solution for $\pi_{n>0}$ is known to have the form $\pi_{n>0} = c\beta^n$, and substituting this assumption we get

$$c\beta^n = c \sum_{i=0}^{\infty} \beta^{n-1+i} r_i \quad \underset{n=1}{\Rightarrow} \quad \beta = \sum_{i=0}^{\infty} \beta^i r_i.$$

Inserting equation 3.38 for r_i we can derive

$$\begin{aligned} \beta &= \sum_{i=0}^{\infty} \beta^i \int_{t=0}^{\infty} \frac{(\mu t)^i}{i!} e^{-\mu t} f_A(t)\,\mathrm{d}t = \int_{t=0}^{\infty} \left(\sum_{i=0}^{\infty} \frac{(\mu\beta t)^i}{i!} \right) e^{-\mu t} f_A(t)\,\mathrm{d}t \\ &= \int_{t=0}^{\infty} e^{\mu\beta t}\, e^{-\mu t} f_A(t)\,\mathrm{d}t = \int_{t=0}^{\infty} e^{-\mu(1-\beta)t} f_A(t)\,\mathrm{d}t = A\big(s{=}\mu(1-\beta)\big) \end{aligned} \tag{3.39}$$

where $A\big(s{=}\mu(1-\beta)\big)$ is the Laplace transform of the arrival distribution evaluated at $s = \mu(1-\beta)$. The trivial solution for $\beta{=}1$ is useless. Luckily, for $\rho < 1$ there always exists another solution to equation 3.39 that falls in the interval $0 < \beta < 1$, which fits to our problem.

The constant c and thereby all π_n we get from $\sum \pi_n = 1$

$$1 = \sum_{i=0}^{\infty} \pi_i = c \sum_{i=0}^{\infty} \beta^i = \frac{c}{1-\beta} \qquad \rightarrow \quad c = 1-\beta \qquad \Rightarrow \quad \pi_n = (1-\beta)\beta^n \tag{3.40}$$

and recognise that the state probabilities π_n at arrival instants are geometrically distributed, similar to the state probabilities we find for the *M/M/1* queue, $p_n = (1-\rho)\,\rho^n$. The global state probabilities of *GI/M/1* are:[4]

$$p_0 = 1-\rho \qquad \text{and} \qquad p_n = \tfrac{\rho}{\beta}(1-\beta)\,\beta^n \tag{3.41}$$

The similarity holds for all system parameters and we could get the relevant performance metrics by substituting some ρ with β in the equations found for *M/M/1*. However, not all ρ are to be replaced, and some ρ may be hidden as $\frac{\lambda}{\mu}$. Thus, dull replacement easily leads to faulty equations.

The flow time T_F of a client arriving when n clients are already waiting equals $n{+}1$ service times S, such that for the Laplace transforms we have $T_F(s|n) = \big(S(s)\big)^{n+1}$. Summing over all n, each weighted with π_n, we get

$$\begin{aligned} T_F(s) = \sum_{n=0}^{\infty} \pi_n \big(S(s)\big)^{n+1} &= \sum_{n=0}^{\infty} (1-\beta)\,\beta^n \left(\frac{\mu}{s+\mu} \right)^{n+1} \\ &= \frac{\mu(1-\beta)}{s+\mu} \sum_{n=0}^{\infty} \left(\frac{\mu\beta}{s+\mu} \right)^n = \frac{\mu(1-\beta)}{s+\mu(1-\beta)} \end{aligned} \tag{3.42}$$

where we used $\sum_{i=0}^{\infty} a^i = \frac{1}{1-a}\ \forall_{a<1}$. The similarity with *M/M/1* is obviouse: replacing β by ρ we get the known flow time distribution.

The *flow and waiting time distributions* found for *GI/M/1* are:

$$T_F(t) = 1 - e^{-\mu(1-\beta)t} \qquad t_F(t) = \mu(1-\beta)\, e^{-\mu(1-\beta)t} \tag{3.43}$$

$$T_W(t) = 1 - \beta\, e^{-\mu(1-\beta)t} \qquad t_W(t) = \beta\mu(1-\beta)\, e^{-\mu(1-\beta)t} \tag{3.44}$$

These are negative exponentially distributed with a step of $1{-}\beta$ at $t{=}0$, which confirms that the geometric sum of negative exponentially distributed random times, here serving times, is itself negative exponentially distributed. The system filling $N(t)$ is not given for its dependence on

[4] a more general derivation of equation 3.41 can be found in [71, section II.3.2 toward equation 3.24]

observation instants. For arrival instants it results from the state probabilities at arrival instants π_i, and for a random observer it results from the global state probabilities p_i.

The mean characteristics of *GI/M/1* result as:[5]

$$\mathrm{E}[N] = \frac{\rho}{1-\beta} \qquad \mathrm{E}[T_F] = \frac{\mathrm{E}[N]}{\lambda} = \frac{1}{\mu(1-\beta)} \qquad \mathrm{E}[T_W] = \mathrm{E}[T_F] - \tfrac{1}{\mu} = \frac{\beta}{\mu(1-\beta)} \tag{3.45}$$

The parameter β depends on the Laplace transform of the arrival distribution, which implicitly comprises all moments. Therefore, we cannot conclude that the mean characteristics depend on few moments only alike it was possible for *M/G/1*. Actually, the systems are in detail quite different.

Using the Laplace transform $A_D(s)=e^{-\frac{s}{\lambda}}$ of *deterministic inter-arrival times* at rate λ, we get from equation 3.39

$$\beta_D = e^{\frac{1-\beta_D}{\rho}}$$

which cannot be solved generic. However, for given ρ it can be numerically approximated.

In figure 3.17 we shows the β that result from equation 3.39 across the feasible load range $0<\rho<1$ for deterministic, phase-type, and Lomax distributed arrivals discussed subsequently. Considering

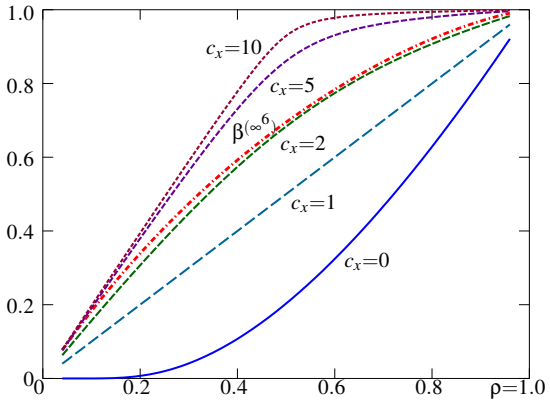


Figure 3.17: Load dependence of the parameter β, required to analyse $GI/M/1$, for increasing coefficient of arrival variation $c_x = [0, 1, 2, 5, 10, \text{ and } \infty^*]$.

the $\beta(c_x{=}0)$ shown in figure 3.17 (the solid line) and the relation to the mean system filling $\mathrm{E}[N_{M/M/1}]$

$$\mathrm{E}\left[N_{GI/M/1}\right] = \frac{\rho}{1-\rho}\,\frac{1-\rho}{1-\beta} = \mathrm{E}\left[N_{M/M/1}\right]\,\frac{1-\rho}{1-\beta} \tag{3.46}$$

we can conclude that for deterministic arrivals the system filling, and thus the system performance, cannot be worse than that for negative exponentially distributed arrivals (straight line $c_x{=}1$) at equal system load ρ because for $\beta \le \rho \Rightarrow \frac{1-\rho}{1-\beta} \le 1\ \forall_{\rho<1}$. This intuitively proves that assuming Markovian inter-arrival times provides an upper bound (worst case) for any smooth and uncorrelated arrival distribution with a standard deviation $\sigma_A \le \frac{1}{\lambda}$.

The opposite applies for bursty arrivals. For the *hyper exponential distribution* (H_n), introduced in section 2.1.2, we achieve a certain coefficient of variation c_x via the H_2 fitting given by equation 2.95 in section 2.3.4 to get the corresponding α_i and λ_i. The Laplace transform of H_2 arrivals is $A_{H_2}(s) = \frac{\alpha_1\lambda_1}{s+\lambda_1} + \frac{\alpha_2\lambda_2}{s+\lambda_2}$, and proceeding as before, inserting $s = \mu(1-\beta_{H_2})$, we get

$$\beta_{H_2} = \frac{\alpha_1\,\lambda_1}{\mu(1-\beta_{H_2})+\lambda_1} + \frac{\alpha_2\,\lambda_2}{\mu(1-\beta_{H_2})+\lambda_2}$$

[5] also to be found in [71, page 210] together with their variances

[6,*] infinite variance is achieved with Lomax distributed arrivals by choosing $1<\alpha\le 2$, here $\alpha{=}2$, the boundary to finite variance. That this causes a less diverging β than H_2 with $c_x>2$ reveals how insufficient c_x characterises a distribution.

a cubic equation in β_{H_2} that can be solved analytically. The root at $\beta=1$ is useless and thus we can divide the cubic equation by $(\beta-1)$ to get a quadratic equation. From the remaining two roots the one in $0<\beta<1$ is the only feasible solution for β_{H_2}. Likewise, we can find the root via the iterative approach used before. If we would not stop the iteration prior machine precision is reached, this is numerically equivalent in terms of achievable calculation precision. To calculate β_{H_2} based on the system load $\rho=\frac{\lambda}{\mu}$ we may set $\mu=1$, such that $\rho=\lambda=(\frac{\alpha_1}{\lambda_1}+\frac{\alpha_2}{\lambda_2})^{-1}$, without loss of generality.

Put to the extreme, heavy tailed *power law* distributed inter-arrival times should cause the worst performance. Here, *Lomax* distributed arrivals, see section 2.2.3 for details on $P_L(\alpha,\sigma)$, with no lower bound and just infinite variance ($\alpha=2$) should comprise a good counterpart to the deterministic arrivals with zero variance. For $P_L(2,\lambda)$ we get the Laplace transform $A_L(s)=2e^{\frac{s}{\lambda}}E_3(\frac{s}{\lambda})$, where $E_n(x)=\int_1^\infty \frac{e^{-xt}}{t^n}\,\mathrm{d}t$ is the generalized exponential integral. Inserting $s=\mu(1-\beta_{P_L})$ in equation 3.39 we get

$$\beta_{P_L}=1-\xi-e^{\xi}\xi^2\left(\ln(\xi)+\frac{\ln(-\frac{1}{\xi})-\ln(-\xi)}{2}+Ei(-\xi)\right)$$

where $\xi=\frac{s}{\lambda}=\frac{1-\beta_{P_L}}{\rho}$, and $Ei(x)=\int_{-\infty}^{x}\frac{e^t}{t}\,\mathrm{d}t$ is the basic exponential integral.[7] This equation has been solved iteratively to get the $\beta^{(\infty)}$ shown in figure 3.17. Obviously, $P_L(2,\lambda)$ is not the worst despite its $\sigma=\infty$, and this reminds us that the coefficient of variation is insufficient to characterise a distribution. Actually, the mathematics in behind these special integral equations are way beyond skills. Still, the simulation results shown in figure 3.18 support the derived analytic result. Anyhow, we will restrain from further analytic treatment of the Lomax and other non-Markovian distributions and use them with simulation studies only.

Figure 3.18 shows the mean system filling $\mathrm{E}[N]$ and flow time $\mathrm{E}[T_F]$ of a $G/M/1$ queueing system for arrivals distributed with increasing coefficient of variation $c_x=[0,1,2,5,10,\text{ and }\infty^*]$. The performance degradation caused by increasing arrival variance differs from what we found for

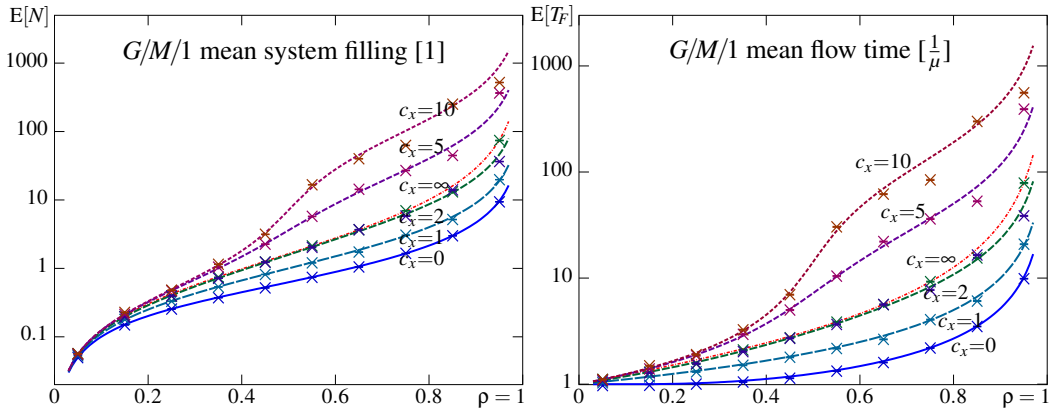


Figure 3.18: Mean system filling $\mathrm{E}[X]$ and flow time $\mathrm{E}[T_F]$ for a $GI/M/1$ queueing system with increasing coefficient of arrival distribution $c_x=[0,1,2,5,10,\text{ and }\infty^*]$.

increasing service time variance with the $M/G/1$ queueing system, shown in figure 3.15. At low loads (i.e., $\rho<0.5$) the degradation is considerably less severe. Only at high loads and for variances above $c_x=2$ we observe an extremely steep rise. The curves for $c_x=[5,10]$ show some unsteadiness, which

[7] The rewriting from the generalized to the basic form of the exponential integral, i.e., $E_n(x)\rightarrow Ei(-x)$, was essential for its calculation with *Octave* and is to be acknowledged to the also freely available computer algebra system *Maxima*. $E_3(x)$ is in *Octave* not available, and an approach via the *incomplete Gamma function* $E_n(x)=x^{n-1}\Gamma(1-n,x)$ failed because its calculation is in *Octave* not supported for the negative half of the Gamma function. In addition, only the normalized version is available and a conversion to the non-normalized by multiplication with $\Gamma(1-n)$ not possible because latter is not defined for negative integer values. Given all these troubles, possible failure could be expected. However, the simulation study approved the analytic result.

results from the increasingly sharp bend that we observe for β_{H_2} in the load range $0.5<\rho<0.6$. It expresses a kind of burst equalising effect that becomes effective when a previous burst of arrivals is with some likelihood not cleared from the queue prior the next burst of arrivals occurs. Above 90% load the system filling reaches infeasible extents.

The simulation results approve the analytic curves in principle; only at high loads and for a coefficient of variation $c_x>2$ the 50.000 samples used to derive simulation results are insufficient to achieve a close match. The simulation routine used here and elsewhere to evaluate queueing systems is presented and discussed in the end of this chapter, section 3.4. The actually used software code can be found in the addenda, section A.I. However, the distribution generators are indirectly called and refer to those presented in chapter 2.

We may conclude that at low mean loads bursty arrivals are a less severe performance killer than bursty service times are. Consequently, measures to smooth arrivals are more important at high loads, while the service time distribution should be kept smooth across the entire load range. Concerning the infinite variance of the *Loamx* distribution that occurs for $1<\alpha\leq 2$ we recognise that the coefficient of variation is insufficient to explain the performance achieved in case of *heavy distributed* arrivals, when $Var(A) = \infty$. A detailed discussion of the theory is well beyond the scope. Still, it might explain why in practice queueing systems commonly perform better than what is predicted based on statistical measurements and traditional models.

3.1.5 *GI/G/1* queueing systems

The most general and versatile approach is to assume nothing upon the distributions that determine a queueing system. However, for compliance we need to assume that arrivals are independent and identically distributed (*GI* not *G*). This most general approach reflects a kind of birth-and-death process where the birth rate is the arrival rate and the death rate is the departure rate. Although not specified, they commonly are correlated via the service process. Even though we do not assume anything we can utilise this correlation to find equations that specify system properties based on the characteristics (moments) of the inter-arrival and service time process.

To analyse *GI/G/1* we need to specify some derived random variables. First the *idle period*, being the random time that the system is idle, and second the random difference between the inter-arrival time and the service time of a customer. The former can be explained graphically, the latter has only procedural meaning and cannot be explained chronologically. Figure 3.19 sketches the time-relation among the different random durations used in queueing theory. The inter-arrival times $T_A(i)$ result

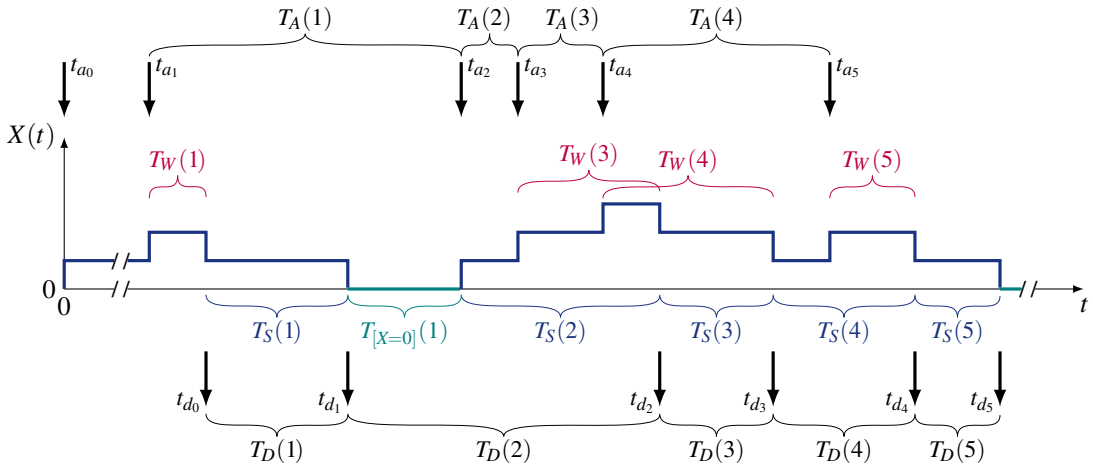


Figure 3.19: Chronology of random durations used in $GI/G/1$ queueing system analysis

from an external process and are assumed to be independent and identically distributed, meaning

time invariant and independent of the system state $X(t)$ and the service times $T_S(i)$. The show $T_S(i)$ intervals specify not only the duration a customer occupies the server, but also when which customer is served (the current server occupation). The *waiting times* $T_W(i)$ are introduced by the queueing system and result from the given *inter-arrival time* T_A and *service time* T_S distribution. The *inter-departure time* T_D describes the distribution of the leaving customers, and thus the arrival process to a downstream system in case of processing cascades.

Let us briefly explain the relations depicted in figure 3.19 based on two adjacent customers: First we consider the case where the first arrives to a non idle system and the second arrives prior the first enters service. Referring to figure 3.19 let us call them customer 3 and 4. The time in between their arrivals is $T_A(3)$. As assumed, customer 4 arrives while the server is busy such that it must be queued, which initialises $T_W(4)$. Waiting ends when serving customer 3 finishes. Finally, $T_S(4)$ tells how long customer 4 occupies the server before it leaves the system. The key is the time overlap of the two adjacent waiting times; it equals $T_W(3) - T_A(3)$ and reveals the recursive relation among waiting times:

$$T_W(4) = T_W(3) + T_S(3) - T_A(3)$$

In case the second customer arrives after serving the first has started but before it leaves, the second again needs to wait for service. In figure 3.19 this case is shown by customers 4 and 5. Their waiting times do not overlap; still, the recursive relation derived for the previous case holds. Here $T_W(4) - T_A(4)$ is negative and expresses the gap between the waiting times of adjacent customers. This equals the time that the first customer has already been served prior the second arrived and thus precisely the amount that needs to be subtracted from its entire service time $T_S(4)$.

Next we consider the case that the second customer arrives after serving of the previous has finished, for example customers 1 and 2 in figure 3.19. Because the second now arrives to an idle system, it does not become queued. Its waiting time $T_W(2)$ is zero. The recursive relation from above would now yield a negative waiting time, which is impossible. However, this negative time enables us to specify the strictly positive *idle period* $T_{[X=0]}$, which only here we call $T_X(1)$ for short, by the relation:

$$0 = T_W(1) - T_A(1) + T_S(1) + T_X(1)$$

This relation holds as well when the first customer also arrives to an idle system: in this case its waiting time is zero and needs to be inserted accordingly ($T_W(1)$=0).

Finally refer customers 2 and 3 in figure 3.19 to the case where the first arrives to an idle system ($T_W(2) = 0$) and the second arrives before the first has left. This case equals the second discussed case where the second customer also arrived while the former is served (customers 4 and 5). The only difference is that in this case the waiting time of the first customer is zero, $T_W(2)$=0, which satisfies the above found relations if inserted accordingly.

Most notably, the above revealed relations do not depend on the properties of the involved processes (A,S). They apply generally, for any arrival and service distribution and for all rates. Consequently, they contribute the basement for the discussion and the derivation of general properties of queueing systems, here and elsewhere.

Waiting time properties of $GI/G/1$ systems

For the *waiting time* $T_W(n)$ we generally find the recursive property that the waiting time of the next arriving client can be expressed by

$$T_W(n+1) = \big(T_W(n) + T_S(n) - T_A(n)\big)_+ = \big(T_W(n) + T_U(n)\big)_+ \tag{3.47}$$

where commonly a new random variable $T_U(n) = T_S(n) - T_A(n)$ is introduced for convenience, $T_A(n)$ is the inter-arrival time in between client n and its successor n+1, and $T_S(n)$ is the service time of

client n. Note that even though $T_U(n)$ is in average negative (for $\lambda<\mu$), it is not strictly negative because it may be positive for some arrival instants.

Introducing another now strictly positive random variable $T_X(n) = -\big(T_W(n) + T_U(n)\big)_-$, expressing the potential time in between the departure of client n and the service begin of client n+1, which only occurs if the system becomes idle prior the arrival of client n+1, we get rid of the non-negativity condition $(\,..\,)_+$.

$$T_W(n+1) - T_X(n) = T_W(n) + T_U(n) \tag{3.48}$$

In steady state is $\mathrm{E}[T_W(n+1)] = \mathrm{E}[T_W(n)]$ and taking the expectations in equation 3.48 we get:

$$\mathrm{E}[T_X] = -\mathrm{E}[T_U] = \frac{1}{\lambda} - \frac{1}{\mu} = \frac{\mu-\lambda}{\lambda\mu} = \frac{1-\rho}{\lambda}$$

Considering that $\mathrm{E}[T_X]$ can as well be calculated from the probability π_0, that system is idle at an arrival instant, times the mean duration it remained idle $\mathrm{E}[T_I]$ until the arrival that found the system idle has occurred, we can calculate latter

$$\mathrm{E}[T_X] = \pi_0\,\mathrm{E}[T_I] \qquad \Rightarrow \qquad \mathrm{E}[T_I] = \frac{\mu-\lambda}{\pi_0\,\lambda\mu} = \frac{1-\rho}{\pi_0\,\lambda} \tag{3.49}$$

henceforth called *mean idle time* $\tau_I = \mathrm{E}[T_I]$. It expresses the mean duration the system remains idle once it became idle, and is not constraint to arrival instants because it is necessarily terminated by any arrival instant and without clients present in the system no departures can occur and thus the system state cannot change during an idle period. Note that from $0\le\rho<1 \leftrightarrow \lambda<\mu$ follows that $\mathrm{E}[T_X] > 0\ \forall_{\rho<1}$ and thus is $\mathrm{E}[T_U] < 0\ \forall_{\rho<1}$.

Taking the squares of both sides of equation 3.48 and taking the expectations of the resultant factors we get

$$\mathrm{E}\big[T_X^2\big] = 2\,\mathrm{E}[T_W]\ \mathrm{E}[T_U] + \mathrm{E}\big[T_U^2\big]$$

where we used $T_W(n+1)\,T_X(n) = 0$ due to their one-sided definitions, independence of $T_W(n)$ and $T_U(n)$, and $\mathrm{E}[T_W(n+1)] = \mathrm{E}[T_W(n)]$ in steady state. Rearranging and inserting $\mathrm{E}\big[T_X^2\big] = \pi_0\,\mathrm{E}\big[T_I^2\big]$ we get a general applicable equation for the *mean waiting time*

$$\mathrm{E}[T_W] = \frac{\pi_0\,\mathrm{E}\big[T_I^2\big] - \mathrm{E}\big[T_U^2\big]}{2\,\mathrm{E}[T_U]} = -\frac{\mathrm{E}\big[T_I^2\big]}{2\,\mathrm{E}[T_I]} - \frac{\mathrm{E}\big[T_U^2\big]}{2\,\mathrm{E}[T_U]} = \frac{Var(T_U) + \mathrm{E}[T_U]^2}{-2\,\mathrm{E}[T_U]} - \frac{\mathrm{E}\big[T_I^2\big]}{2\,\mathrm{E}[T_I]} \tag{3.50}$$

using that $\mathrm{E}[T_X] = -\mathrm{E}[T_U]$ and thus $\mathrm{E}[T_U] = -\pi_0\,\mathrm{E}[T_I]$, and finally $\mathrm{E}[T_U^2] = Var(T_U) + \mathrm{E}[T_U]^2$. We presume independence of T_S and T_A and because $T_U = T_S - T_A$ we have $-\mathrm{E}[T_U] = \frac{1}{\lambda} - \frac{1}{\mu} = \frac{1-\rho}{\lambda}$ and $Var(T_U) = \sigma_A^2 + \sigma_B^2$. Inserting these we get after some rearrangements:

$$\mathrm{E}[T_W] = \frac{\lambda(\sigma_A^2 + \sigma_B^2)}{2(1-\rho)} + \frac{1-\rho}{2\lambda} - \frac{\mathrm{E}\big[T_I^2\big]}{2\,\mathrm{E}[T_I]} \tag{3.51}$$

The result comprises the first and second moments of the arrival and the service distribution and halve the relation of the second and first raw moment of the idle time. Latter needs to be derived for the actual distributions involved, which in some cases may be intractable.

Upper and lower bounds for the mean waiting time can be derived considering signs and relations of the components involved in equation 3.50 such that we finally get

$$\begin{gathered}\frac{\lambda^2\sigma_B^2 - 2\rho + \rho^2}{2\lambda(1-\rho)} \le \mathrm{E}[T_W] \le \frac{\lambda\left(\sigma_A^2 + \sigma_B^2\right)}{2(1-\rho)} \\ \left(\frac{c_{x,B}^2}{2} - \frac{2-\rho}{\rho}\right)_+ \frac{\rho}{1-\rho}\,\frac{1}{\mu} \le \mathrm{E}[T_W] \le \frac{(\frac{c_{x,A}}{\lambda})^2 + (\frac{c_{x,B}}{\mu})^2}{2}\,\frac{\rho}{1-\rho}\,\frac{1}{\mu}\end{gathered} \tag{3.52}$$

where the terms appearing on both sides state the mean waiting time of the *M/M/1* queueing system (equation 1.37). To state the bounds such that the coefficient of variation c_x replaces the standard deviation σ we use $c_{x,A} = \lambda\sigma_A$ and $c_{x,B} = \mu\sigma_B$ such that $\sigma_A^2 + \sigma_B^2 = (\frac{c_{x,A}}{\lambda})^2 + (\frac{c_{x,B}}{\mu})^2$. Also noteworthy, the lower bound does not depend on the second moment of the arrival distribution while the upper bound considers both. This again indicates that the service variance has more impact on the performance than the arrival variance, comparable to the conclusions we have drawn previously comparing the performance of $M/G/1$ and $GI/M/1$ for increasing variation indices.

It would be nice to present the bounds, particularly the gap in between them, for increasing variation indices and load in a single figure. However, having three variables we would need to present the gap in between two four dimensional surfaces, which is impossible in two dimensions. Therefore, figure 3.20 shows a set of four sub-figures, each presenting a different cut through the bounding surfaces. Also shown is the approximation presented and discussed shortly.

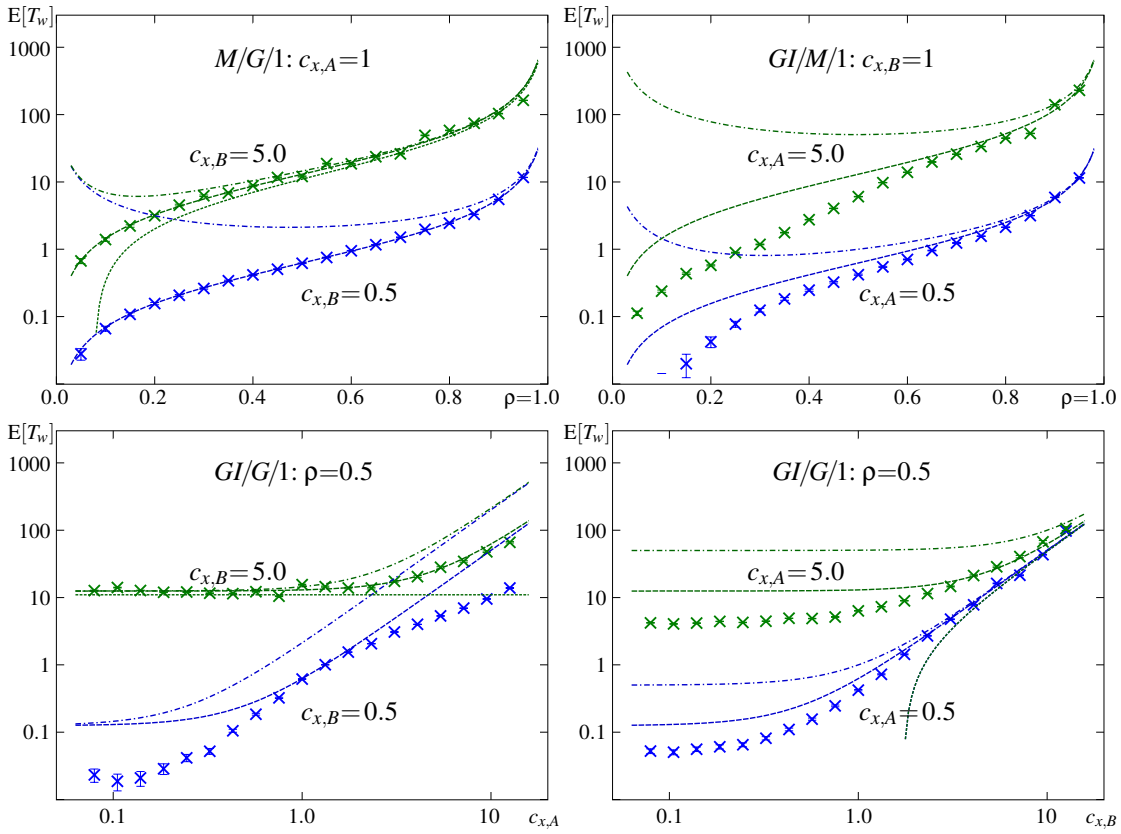


Figure 3.20: Simulated $GI/G/1$ waiting time $\overline{T_w}$=E[T_w] samples (×) combined with its calculated approximation $\widehat{T_w}$ (---) and lower/upper bounds $\lfloor T_w \rceil$ (···,-·-) all normalised to μ=1 ↔ λ=ρ.

We use 100.000 samples per point to achieve a sufficient match. Still, some fluctuation of the statistical reaults remains visible, particularly for high c_x at high loads. For low service variances the lower bound is zero, thus not visible with logarithmic scaling. The gap between upper and lower bound can be huge, and the bounds therefore rather wide apart from the actual performance. The simulated samples $\overline{T_w}$ approach the upper bound at high loads only, which enables simplified mathematical treatment of high load situations. The opposite can be said about low load regions. Still, these regions comprise the common load situations of well managed packet switching systems, including MPLS architectures. The divergence of samples from the approximation is discussed after latter has been formally introduce by equation 3.53.

Finally, using the above stated bounds for $GI/G/1$ and comparing these with the results found for $M/G/1$ and $GI/M/1$ a utile *approximation* for the mean waiting time has been defined by John

Kingman in [72], today known as *Kingman's equation*

$$\begin{aligned}\widehat{T_W}(GI/G/1) &= \frac{\rho\left(\sigma_A^2\lambda^2+\sigma_B^2\mu^2\right)}{2\mu(1-\rho)}\\ &= \frac{c_{x,A}^2+c_{x,B}^2}{2}\,\frac{\rho}{1-\rho}\,\frac{1}{\mu} = \frac{c_{x,A}^2+c_{x,B}^2}{2}\,\mathrm{E}[T_W(M/M/1)]\end{aligned} \tag{3.53}$$

where $c_{x,A}$, $c_{x,B}$ are the coefficient of variation of the inter-arrival time and service time distribution, respectively ($c_{x,i}$=$\sigma_i\mu_i$). The approximation is exact for $M/M/1$, where $c_{x,A}^2$=$c_{x,B}^2$=1, as well as for $M/G/1$, where $c_{x,A}^2$=1.

Returning to figure 3.20 we recognise that, evident from the equation above, the approximation is exact if the arrival process is Markovian, and that the approximation is the same for swapped variances, presented side by side in left and right sub-figures. This results from the symmetry of the approximation (equation 3.53), where the arrival and service variance is equally considered. In real, as shown by the simulated examples $\overline{T_w}$, there is no symmetry and the actual performance differs depending on the relation $c_{x,A}$ to $c_{x,B}$. However, all sub-figures show $\overline{T_w}\leq\widehat{T_w}$, indicating that the approximation tends to overestimate the actual waiting time and thus, $\widehat{T_w}$ seems to provide a weak (likely) upper bound. Here, we have to note that for the simulation we applied $Erlang_k$ and *Cox* fitting for $c_x^2 < 0.5$ and $c_x^2 \geq 0.5$ respectively. The thereby implicitly chosen higher moments may influence the simulation results, such that $\overline{T_w} \leq \widehat{T_w}$ needs to be approved case by case if distributions are explicitly specified. However, the bounds and approximations comprise first and second moments only, indicating that higher moments should impact the performance rather weakly.

GI/G/n queueing systems

Completing the infinite queueing systems where actually queueing occurs, see section 3.3 for other infinite systems, we briefly sketch the results for *GI/G/n* systems where multiple servers are available. Note that the systems considered here are restricted to identical servers, meaning servers that do not differ in their performance. Many communication systems rely physically on paralleled resources, which could be modelled as multi-server system. However, some autonomic and to the traffic load fully transparent mechanism typically joins these parallel resources in a way that yields the combined capacity so efficiently that for analysing the layer above we can assume a single server providing the joint capacity. For example, a 16-bit wide bus that transports a continuous bit-streams in parallel causes a latency that is identical to that of a 16-times faster serial connection. Both can be modelled by a single server queueing system, even though its structure only matches the latter. Therefore, we do not put much emphasis on the *GI/G/n* theory and skip any mathematical derivation. In particular because the available bounds and approximations for the waiting time, $\lfloor T_w \rceil$ and $\widehat{T_w}$, are anyhow developed based on the results found for *GI/G/1*.[8]

$$\left(\frac{\lambda^2\left(\frac{\sigma_B}{n}\right)^2-2\varrho+\varrho^2}{2\lambda(1-\varrho)}-\frac{n-1}{2}\left(\frac{\mu}{n}\sigma_B^2+\varrho\right)\right)_+ \leq \mathrm{E}[T_W] \leq \frac{\lambda\left(\sigma_A^2+\left(\frac{\sigma_B}{n}\right)^2\right)}{2\,(1-\varrho)} \tag{3.54}$$

Similar to the $GI/G/1$ approximation for the waiting time $\widehat{T_w}$, which can be based on the precisely known $\mathrm{E}[T_w(M/M/1)]$, an approximation for $\widehat{T_w}(GI/G/n)$ can be stated based on the exact result

[8]The derivation is presented in [57] and refers to the works of Kingman, Brumelle, and De Smit.

known for $\mathrm{E}[T_w(M/M/n)]$, given in equation 3.15. Doing so, we get

$$\begin{aligned}\widehat{T_W}(GI/G/n) &= \frac{c_{x,A}^2 + c_{x,B}^2}{2}\,\mathrm{E}[T_W(M/M/n)] \\ &= \frac{c_{x,A}^2 + c_{x,B}^2}{2}\,\frac{\rho^n p(0)}{n!(1-\varrho)^2}\,\frac{1}{n\mu} \\ &= \frac{c_{x,A}^2 + c_{x,B}^2}{2}\,\frac{p(i\geq n)}{\lambda}\,\frac{\varrho}{1-\varrho}\end{aligned} \tag{3.55}$$

where $p(0)$ is the probability that the system is idle given by equation 3.12, and $p(i\geq n)$ is the probability that all n servers are busy, given by equation 3.13. Note that $\varrho = \frac{\lambda}{n\mu}$ is the system load, whereas $\rho = \frac{\lambda}{\mu}$ states the service time to inter-arrival time relation, where only for latter the *Erlang* unit is commonly used. This utile approximation (equation 3.55) is commonly refereed to as the *Allen-Cunneen approximation* widely used for its immanent simplicity. Several improved approximations based on this one can be found in the literature, see for example the ones noted in [26, p.266]. These mostly introduce some case dependent correction factor in order to achieve a better approximation, but no fundamentally different approach.

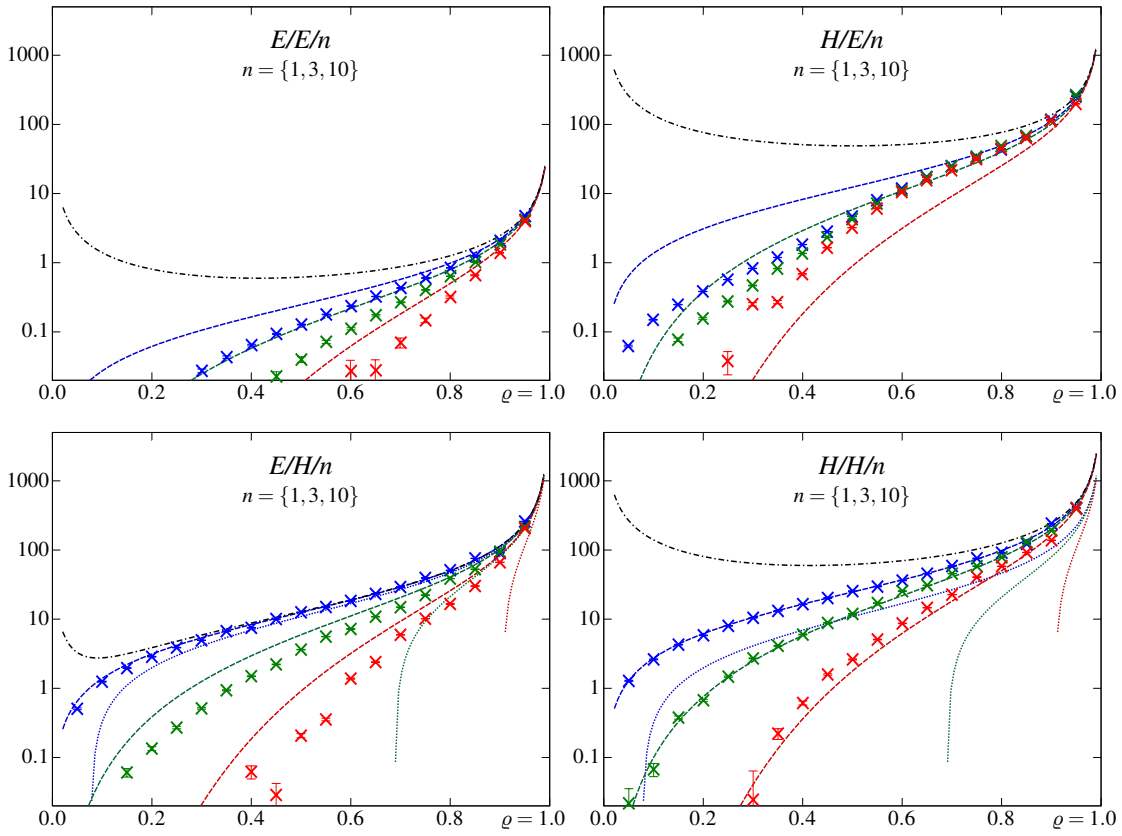


Figure 3.21: Waiting time of $GI/G/n$ systems for increasing number of servers n=$\{1,3,10\}$ over load for different combinations of inter-arrival and service time variances ($c_{x,i}$=$[0.5,5]$), including T_W bounds (lower *dotted*, upper *dash-dotted*), approximation $\widehat{T_W}$ (*dashed*), and simulation results.

The simulation results shown in figure 3.21 prove that for one server and hyper-exponential service times the approximation fits. Quite surprisingly it does not so for smooth service times. For smooth arrivals and smooth service it seems to be rather pessimistic, but for bursty arrivals and smooth service it fails to provide guidance. However, the bounds hold. Still, for smooth service the lower bound is zero, and the upper bound is in any case independent of the number of servers, such

that the bounds rarely define a tight region. In addition we recognise that the gap between upper and lower bound increases with the number of servers. For many servers only the approximation provides a useful hint on the expectable performance. Latter fits considerably better for bursty service times, and seems not very appropriate for smooth distributed processes; it could be an upper bound in case of smooth inter-arrival times, but for smooth holding times the approximation yields results above as well as below the true value.

3.1.6 The matrix geometric approach to infinite queues

This method has been introduced by Marcel F. Neuts [73, 1981] and utilises the strengths of numeric matrix computation by automated calculation machines (computers). It applies the *matrix analytic method* applicable for finite queueing systems introduced in chapter 1 section 1.4 and presented in detail in the following section 3.2.3, to infinite systems. To use it, *phase-type distributed* inter-arrival and service processes are required, at least an approximate model based on Markov phases. The key is to find a repetitive state transition structure of the form shown in figure 3.22, which else cannot be achieved. The structure comprises a single *border level* including the idle state, and identical

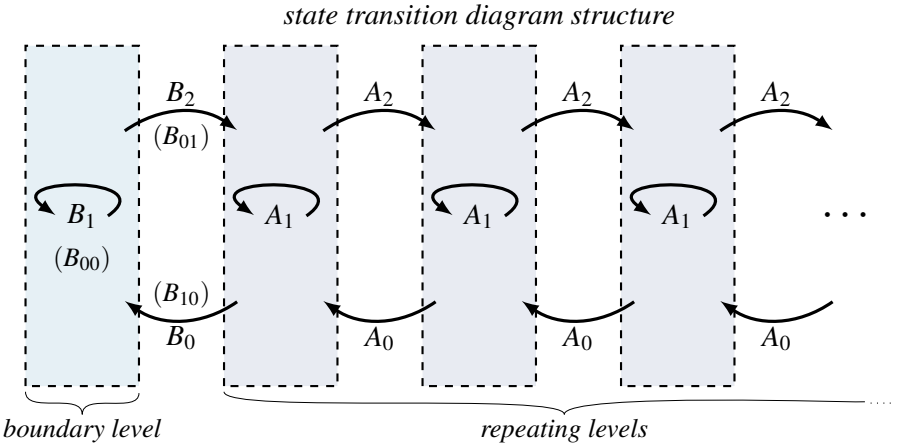


Figure 3.22: State transition structure required for the matrix geometric approach

repeating levels to its right. Transitions have to be restricted to adjacent levels, such that there are only *boundary transitions* between the border level and the first repeating level, and identical transitions between any two adjacent repeating levels.

Given this structure, we can specify six matrices that hold the level internal, upward and downward transitions such that all possible transitions are covered. Commonly the matrices related to the border level are referred to by B_{00}, B_{01}, B_{10}, where the indices tell the transitions orientation. The matrices related to repeating levels are differently indexed; here A_1 refers to internal, A_2 to upward, and A_0 to downward. In the following we remain with the letters but use the A indexing also for Bs, as shown in figure 3.22.

The resultant block transition rate matrix Q has the form

$$Q = \begin{bmatrix} B_1 & B_2 & 0 & 0 & 0 & 0 & \cdots \\ B_0 & A_1 & A_2 & 0 & 0 & 0 & \cdots \\ 0 & A_0 & A_1 & A_2 & 0 & 0 & \cdots \\ 0 & 0 & A_0 & A_1 & A_2 & 0 & \cdots \\ \vdots & \vdots & & \ddots & \ddots & \ddots & \end{bmatrix} \tag{3.56}$$

and to get the state probability vector π we need to solve

$$\pi Q = 0 \tag{3.57}$$

which is quite a challenge given the infinite number of dimensions. However, due to the repetitive structure relate all level-wise sub-vectors $\pi_{i>1}$ to π_1:

$$\pi_i = \pi_1 R^{i-1} \tag{3.58}$$

The matrix R therein is Neuts' rate-matrix, which can be calculated numerically only, using the quadratic relation $A_2 + RA_1 + R^2A_0 = 0$. For example, by successive substitution

$$R_{(k+1)} = -A_2A_1^{-1} - R_k^2A_0A_1^{-1} \tag{3.59}$$

as proposed by Neuts, or more efficiently by the *logarithmic reduction algorithm* proposed by Latouche and Ramaswami in [74]. Latter shows quadratic convergence (doubled precision per iteration) and is thus the state-of-the-art method commonly used.

Using $\pi_2 = R\pi_1$ we can reduce equation 3.57 to:

$$\begin{pmatrix} \pi_0 & \pi_1 \end{pmatrix} \begin{bmatrix} B_1 & B_2 \\ B_0 & A_1 + RA_0 \end{bmatrix} = \begin{pmatrix} 0 & 0 \end{pmatrix} \tag{3.60}$$

Solving this to get π_0 and π_1, using the method presented in chapter 1 section 1.4.3, we can recursively calculate $\pi_i \; \forall_{i>1}$ via equation 3.58. Finally, the achieved state vector elements need to be normalised to $\sum \pi = 1$, which is achieved by the scaling

$$\pi_i \;\leftarrow\; \frac{\pi_i}{\sum \pi_0 + \sum\left(\pi_1 (I-R)^{-1}\right)} \tag{3.61}$$

where $\sum x$ refers to the scalar sum over the dimensional elements x_i of the vector $\vec{x}$, mathematically being the multiplication of the vector $\vec{x}$ with the dimensions-vector $\vec{e}$ (ones vector), i.e., $\sum x = \vec{x}\vec{e}$, and I is the identity matrix (ones diagonal).

To calculate the common performance metrics it is not necessary to calculate the infinitely many state probability vectors π_i per level. The mean system filling $\mathrm{E}[X]$ can be calculated directly from Neuts' rate matrix R and the first two state probability vectors π_0 and π_1, which we get from equation 3.60. However, the often referenced equation 3.62

$$\mathrm{E}[X] = \sum_{i=0}^{\infty} \sum_j i\, p_j(x{=}i) = \cdots = \left\| \pi_1 (I-R)^{-2} \right\|_1 \tag{3.62}$$

is applicable only for true *QBD*s; meaning that the level index i has to equal the number of clients in the system, which is possible only if the levels can be defined accordingly and implies that the boundary level comprises idle states only. This restriction limits the usability of this equation, but not the applicability of the matrix geometric method per se. If a *QBD* compliant level definition is not possible, the calculation of $\mathrm{E}[X]$ has to consider the actual level design such that the calculation actually implements the summation on the left side of the ellipsis in equation 3.62.

Once $\mathrm{E}[X]$ is calculated the other mean metrics follow from Little's law $N{=}\lambda T$ and $T_f{=}T_w{+}T_S$.

$$\mathrm{E}[T_f] = \frac{\mathrm{E}[X]}{\lambda_g} \qquad \mathrm{E}[T_w] = \mathrm{E}[T_f] - \frac{1}{\mu_S} = \frac{\mathrm{E}[X] - n\,\varrho}{\lambda_g} \qquad \mathrm{E}[Q] = \lambda_g \mathrm{E}[T_W] = \mathrm{E}[X] - n\,\varrho \tag{3.63}$$

Here, μ_S is the rate provided by a single server, which may be composed by several phases with a particular μ_i each. To clearly separate gross and individual rates we use the index g to expresses the gross system relation. This is essentially required for multi server systems with their total service rate $\mu_g = n\mu_S$, but as well for arrival processes that are composed from several phases with rates λ_i.

The *matrix geometric approach* is very powerful, particularly when it comes to approximate solutions comprising *spectral components* that substitute a distribution by a combination of negative

exponentially distributed components, resulting in *phase-type* process models. Using phase-type distributions any distribution can be approximated as accurate as wished, though that may require many or even infinitely many phases. In this context we should note that there also exist solutions to problems where the transition matrix structure is of upper or lower Hessenberg type [34, 73]. Their solution is mathematically more strenuous but also more efficient because the matrix size remains considerably smaller if the transition matrix can be reduced to such a structure.

Example $H_2/E_2/2$-system analysis using the Matrix geometric method

To sketch how this method is applied, we briefly go through an example where the inter-arrival times are hyper-exponential H_2 distributed, the service times are hypo-exponential with E_2 (Erlang-2) and where we have two indentical servers wotking in parallel. To correctly draw the state transition diagram we need to consider the contributing phase-type processes first. In figure 3.23 we sketch how the structure is composed based on the structures of the two processes. The state indices comprise

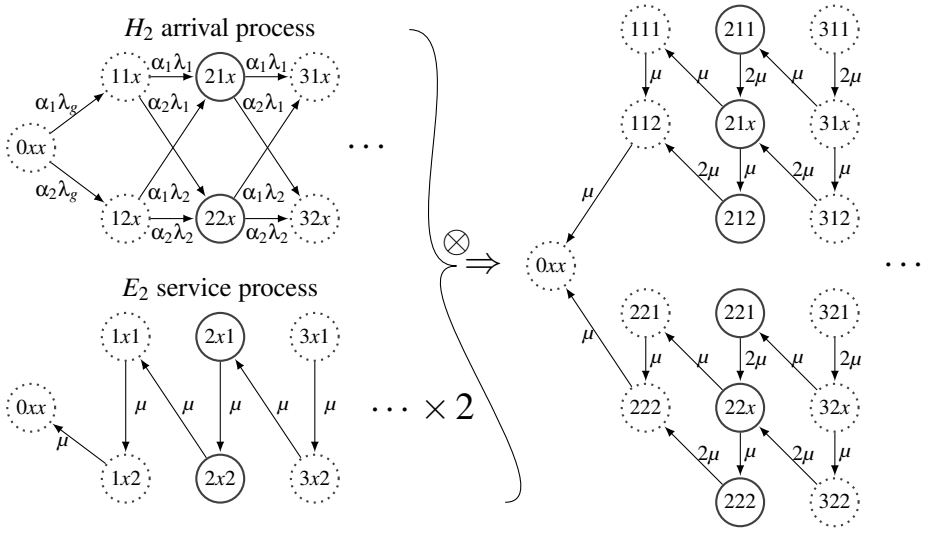


Figure 3.23: The H_2 arrival and $2\times E_2$ service process phases

three digits, the first tells the number of customers in the system (X), the second the phase of the arrival process (λ_1 or λ_2), and the third tells the phase of the departure process (1^{st} or 2^{nd} step). An x is used in the index if it is undetermined. In the idle state the arrival rate is the global arrival rate of the H_2 process $\lambda_g = (\frac{\alpha_1}{\lambda_1} + \frac{\alpha_2}{\lambda_2})^{-1}$. If the process is too complex to calculate the gross rate λ_g we can model it by multiple idle phases. The transitions among these idle phases than are the phase probabilities α_i and the upward transitions are $\alpha_i\lambda_i$ alike those between levels. For details and many more examples on how to compose and solve state transition diagrams for phase type distributions please refer to the doctoral thesis of Markus Sommereder and his book [75, 76].

The combined state transition diagram on the right in figure 3.23 shows only the service transitions to highlight how the two parallel servers are considered. Servers working in parallel contribute each its share to the service rate and only because phases contribute negative exponential and thus *memoryless* holding times, we do not need to consider which servers is in which phase. In a time continuous regime the one that finishes first triggers the transition, thus there can be only a single state change at a time. However, we have to consider all possible mixes of server phases that may occur at any time. Therefore, we need in our example one additional state in the service chain, where one server is in phase-1 and the other in phase-2. Here we index this by an x to express that the server group is not in the same phase. For more servers we would need accordingly more additional phases, and for higher order Ph_k-process models also accordingly more phases exist, causing a binomial rising number of possible mixtures to be considered as dedicated states each, $m_S = \binom{n+k-1}{n}$, where n

is the number of parallel identical servers and k the number or service phases per server. The number of arrival related sub-states per level in case of parallel arriving flows could be calculated similarly if the arrivals are identically distributed. However, if arrival flows are individually distributed, than the phases need to be considered per flow and we get $m_A = \prod_{i=1}^{n} k_i$ as the number of arrival sub-states, where n is the number of arrival flows and k_i the number of arrival phases of flow i. The total *number of sub-states per system level* is the multiplication of the arrival and server sub-spaces:

$$m = m_A\, m_S \quad \text{with} \quad m_A = \prod_{i=1}^{n_A} k_A(i) \quad \text{and} \quad m_S = \binom{n_S + k_S - 1}{n_S} = \frac{(n_S + k_S - 1)!}{n_S!\,(k_S - 1)!} \tag{3.64}$$

In figure 3.23 we notice that grouping the phases in columns where per column the number of customers is the same, we get a *QBD* like structure where transitions among distant columns do not exist. All transitions are within a column or among adjacent columns. Next we recognise that for columns to the right of the shown level (X=2), including itself, the phases and the transition rates are the same, independent of the number of customers in the system. This is exactly what we need to apply the matrix geometric method. Figure 3.24 shows the complete transition diagram with the level assignment. For clarity the transition rates are only sparsely stated; please refer to figure 3.23 and

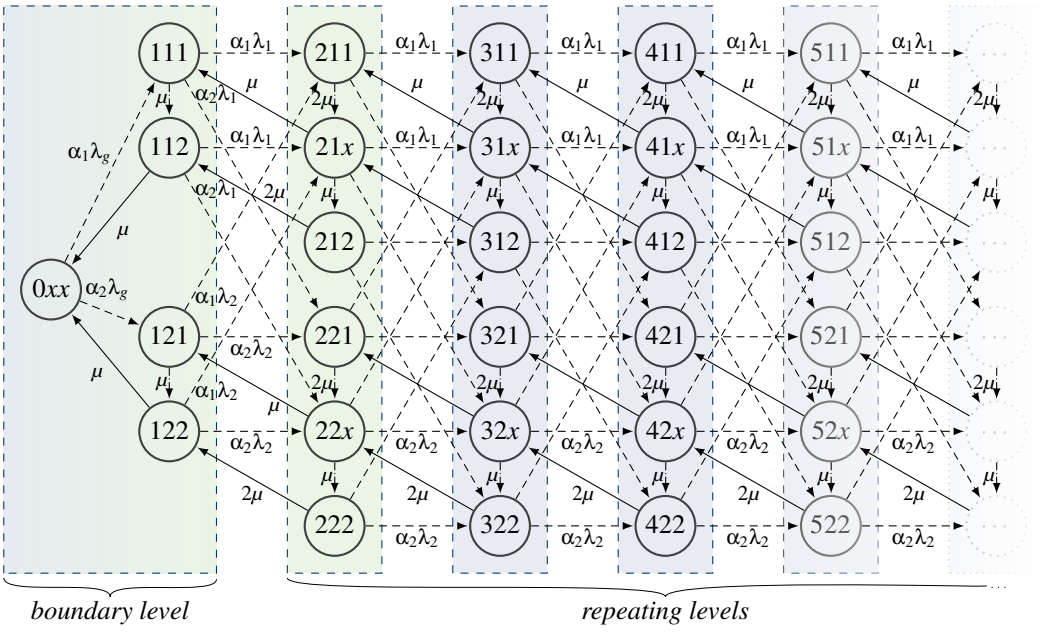


Figure 3.24: $H_2/E_2/2$ state transition diagram with level assignment

figure 3.25 for those not shown. The transitions are drawn in different line styles: full, dash-dotted, and dashed; because we need to consider downward, internal, and upward transitions separately.

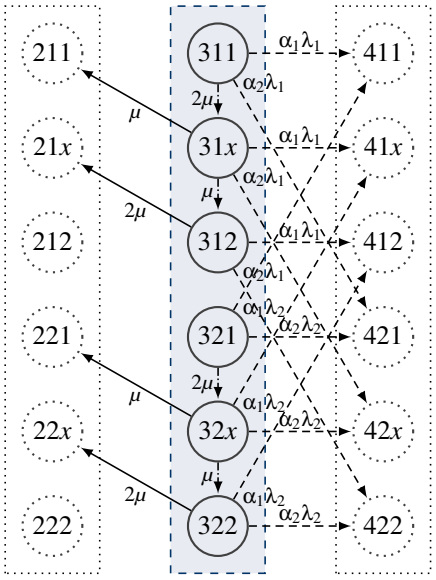
The *boundary level* covers the idle state and all states where the applied service rate is state dependently less than the maximum, here 2μ. The border transitions between the boundary and the first repeating level fit to the different numbers of sub-states contained in boundary and repeating levels. For the boundary related B-matrices we have

$$B_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 & 0 \\ 0 & 0 & 2\mu & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 2\mu \end{bmatrix} \quad B_1 = \begin{bmatrix} * & \alpha_1\lambda_g & 0 & \alpha_2\lambda_g & 0 \\ 0 & * & \mu & 0 & 0 \\ \mu & 0 & * & 0 & 0 \\ 0 & 0 & 0 & * & \mu \\ \mu & 0 & 0 & 0 & * \end{bmatrix} \quad B_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \alpha_1\lambda_1 & 0 & 0 & \alpha_2\lambda_1 & 0 & 0 \\ 0 & \alpha_1\lambda_1 & 0 & 0 & \alpha_2\lambda_1 & 0 \\ \alpha_1\lambda_2 & 0 & 0 & \alpha_2\lambda_2 & 0 & 0 \\ 0 & \alpha_1\lambda_2 & 0 & 0 & \alpha_2\lambda_2 & 0 \end{bmatrix} \tag{3.65}$$

where $\lambda_g = (\frac{\alpha_1}{\lambda_1} + \frac{\alpha_2}{\lambda_2})^{-1}$. The diagonal elements $[*]$ in B_1 can be calculated from the others:

$$\begin{aligned} &B_g \times 1 = 0 \quad \text{where} \quad B_g = B_1 + B_2 \quad \text{and 1 is the unit vector} \\ &\text{such that} \quad -b_{i,i}(B_1) = \sum_{j \neq i} b_{i,j}(B_1) + \sum_j b_{i,j}(B_2) \quad \rightarrow \quad b_{i,i} = \begin{pmatrix} -\lambda_g \\ -\lambda_1 - \mu \\ -\lambda_1 - \mu \\ -\lambda_2 - \mu \\ -\lambda_2 - \mu \end{pmatrix} \end{aligned} \tag{3.66}$$

The inter-level transitions in between repeating levels are a little too crowded in figure 3.24 to state all rates. Therefore, figure 3.25 shows a single repeating level with its *downward*, *internal*, and *upward* transitions together with the corresponding matrices A_0, A_1, and A_2. The diagonal



$$A_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \mu & 0 & 0 & 0 & 0 & 0 \\ 0 & 2\mu & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu & 0 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} * & 2\mu & 0 & 0 & 0 & 0 \\ 0 & * & \mu & 0 & 0 & 0 \\ 0 & 0 & * & 0 & 0 & 0 \\ 0 & 0 & 0 & * & 2\mu & 0 \\ 0 & 0 & 0 & 0 & * & \mu \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix}$$

$$A_2 = \begin{bmatrix} \alpha_1\lambda_1 & 0 & 0 & \alpha_2\lambda_1 & 0 & 0 \\ 0 & \alpha_1\lambda_1 & 0 & 0 & \alpha_2\lambda_1 & 0 \\ 0 & 0 & \alpha_1\lambda_1 & 0 & 0 & \alpha_2\lambda_1 \\ \alpha_1\lambda_2 & 0 & 0 & \alpha_2\lambda_2 & 0 & 0 \\ 0 & \alpha_1\lambda_2 & 0 & 0 & \alpha_2\lambda_2 & 0 \\ 0 & 0 & \alpha_1\lambda_2 & 0 & 0 & \alpha_2\lambda_2 \end{bmatrix}$$

Figure 3.25: $H_2/E_2/2$ repeating level with transitions and A-matrices thereof

elements $[*]$ in A_1 can be calculated from the others:

$$\begin{aligned} &A_g \times 1 = 0 \quad \text{where} \quad A_g = A_0 + A_1 + A_2 \quad \text{and 1 is the unit vector} \\ &\text{such that} \quad -a_{i,i}(A_1) = \sum_j a_{i,j}(A_0) + \sum_{j \neq i} a_{i,j}(A_1) + \sum_j a_{i,j}(A_2) \quad \rightarrow \quad a_{i,i} = \begin{pmatrix} -\lambda_1 - 2\mu \\ -\lambda_1 - 2\mu \\ -\lambda_1 - 2\mu \\ -\lambda_2 - 2\mu \\ -\lambda_2 - 2\mu \\ -\lambda_2 - 2\mu \end{pmatrix} \end{aligned} \tag{3.67}$$

With the manually filled matrices we can calculate the *rate matrix R* using equation 3.59 and solve the queueing system by calculating the state probabilities using equation 3.60 and 3.61 or directly the performance metrics using equation 3.63 once the mean system filling $\mathrm{E}[X]$ is known.

However, defining these matrices by evaluating a drawn state transition diagram can be cumbersome and error-prone. Luckily, the matrices that define the system can be systematically achieved directly from the matrices defining the involved phase-type processes. For A and S we have

$$Q_A[H_2] = \begin{bmatrix} -\lambda_1 & 0 \\ 0 & -\lambda_2 \end{bmatrix} + \begin{bmatrix} \alpha_1\lambda_1 & \alpha_2\lambda_1 \\ \alpha_1\lambda_2 & \alpha_2\lambda_2 \end{bmatrix} = \begin{bmatrix} -\alpha_2\lambda_1 & \alpha_2\lambda_1 \\ \alpha_1\lambda_2 & -\alpha_1\lambda_2 \end{bmatrix} \tag{3.68}$$

$$Q_S[2\times E_2] = \begin{bmatrix} -2\mu & 2\mu & 0 \\ 0 & -2\mu & \mu \\ 0 & 0 & -2\mu \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ \mu & 0 & 0 \\ 0 & 2\mu & 0 \end{bmatrix} = \begin{bmatrix} -2\mu & 2\mu & 0 \\ \mu & -2\mu & \mu \\ 0 & 2\mu & -2\mu \end{bmatrix} \tag{3.69}$$

where we use the *MAP* approach, $Q = D_0 + D_1$ as presented in section 2.1.3, with D_0 consisting of the non-event-generating transitions and D_1 covering all event-generating transitions of the processes shown in figure 3.23. For the hyper-exponential process H_n we get the event generating sub-matrix D_1 from $D_1[H_n] = \lambda \times \alpha$, and $D_0[H_n] = diag(-\lambda)$ is a diagonal matrix comprising the negative rate vector elements λ_i in its diagonal. We note that for a precise phase-type definition the *rates of phases* need to be specified individually. The gross rates λ_g or μ_g result uniquely from the rates of phases and the transitions among phases; vice versa the relation is under-determined and thus indefinite.

Given the matrices of the arrival and service process, particularly the splitting in D_0 and D_1, we can calculate the sub-matrices required to solve the system using the matrix geometric approach according to

$$A_0 = I_{m_A} \otimes D_1(S) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 0 \\ \mu & 0 & 0 \\ 0 & 2\mu & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \mu & 0 & 0 & 0 & 0 & 0 \\ 0 & 2\mu & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu & 0 \end{bmatrix} \tag{3.70}$$

$$A_1 = D_0(A) \otimes I_{m_S} + I_{m_A} \otimes D_0(S) = \begin{bmatrix} -\lambda_1-2\mu & 2\mu & 0 & 0 & 0 & 0 \\ 0 & -\lambda_1-2\mu & \mu & 0 & 0 & 0 \\ 0 & 0 & -\lambda_1-2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & -\lambda_2-2\mu & 2\mu & 0 \\ 0 & 0 & 0 & 0 & -\lambda_2-2\mu & \mu \\ 0 & 0 & 0 & 0 & 0 & -\lambda_2-2\mu \end{bmatrix} \tag{3.71}$$

$$A_2 = D_1(A) \otimes I_{m_S} = \begin{bmatrix} \alpha_1\lambda_1 & \alpha_2\lambda_1 \\ \alpha_1\lambda_2 & \alpha_2\lambda_2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha_1\lambda_1 & 0 & 0 & \alpha_2\lambda_1 & 0 & 0 \\ 0 & \alpha_1\lambda_1 & 0 & 0 & \alpha_2\lambda_1 & 0 \\ 0 & 0 & \alpha_1\lambda_1 & 0 & 0 & \alpha_2\lambda_1 \\ \alpha_1\lambda_2 & 0 & 0 & \alpha_2\lambda_2 & 0 & 0 \\ 0 & \alpha_1\lambda_2 & 0 & 0 & \alpha_2\lambda_2 & 0 \\ 0 & 0 & \alpha_1\lambda_2 & 0 & 0 & \alpha_2\lambda_2 \end{bmatrix} \tag{3.72}$$

where $\otimes$ refers to the Kronecker (tensor) multiplication, explicitly shown in equation 3.70 and 3.72. The I_{m_X} is the identity matrix, also called ones diagonal or eye-matrix, of size m_X, as also explicitly shown in 3.70 and 3.72. The size m_X equals the order of the other process, also called rank and related to the number of process phases, as for example calculated in equation 3.64. The negative diagonal elements of the intra-level transition matrix A_1 result straight from applying the generator matrices, and need not be calculated separately, as shown in equation 3.71.

The border transitions B_0, B_2 and the boundary-level internal transitions B_1 are usually easy to identify in the state transition diagram. However, for true *QBD*s with absolutely identical levels except the idle state, these also can be calculated from the generator matrices of the involved processes [34, 76]. Here, for the multi server example, we constructed a *phase-type* service process to replace the two individual servers. This constructed service process is valid for repeating levels only because it postulates two busy servers. Consequently, $Q_S[2\times E_2]$ is not applicable in the boundary level and thus, it cannot be used to calculate the transition matrices B_0, B_1, B_2.

Numeric results

The recursive calculation of the rate matrix requires numeric values. To calculate it we have to know λ_1, λ_2, α_1, α_2, and μ. To complete the example we assume for a start the following values:

$$\lambda = \begin{pmatrix} 0.5 \\ 2.0 \end{pmatrix} \qquad \alpha = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix} \qquad \mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{3.73}$$

Note that in this example setting the Erlang phase's μ=1 results in a normalized system with capacity μ_g=1. Due to the sequential phases of the E_k-process each server contributes a capacity of $\mu_S = \frac{\mu}{k}$ only. If the number of servers equals the order of the Erlang process (n=k), as in our example, than they achieve in total the intended normalised capacity $\mu_g = n\mu_S = 1$ for μ=1. Else, μ needs to be chosen accordingly in order to achieve normalised queueing systems that can be compared. For the above chosen values the resultant system load $\varrho = \frac{1}{\mu_g}(\frac{\alpha_1}{\lambda_1} + \frac{\alpha_2}{\lambda_2})^{-1} < 1$ grants stability and we get

$$R = \begin{bmatrix} 0.2269727 & 0.2583543 & 0.1033417 & 0.0369539 & 0.0233078 & 0.0058270 \\ 0.0499852 & 0.2463287 & 0.0985315 & 0.0043595 & 0.0377387 & 0.0094347 \\ 0.0211675 & 0.0710474 & 0.1784190 & 0.0015686 & 0.0064096 & 0.0328524 \\ 0.9078909 & 1.0334172 & 0.4133669 & 0.1478156 & 0.0932312 & 0.0233078 \\ 0.1999407 & 0.9853148 & 0.3941259 & 0.0174379 & 0.1509548 & 0.0377387 \\ 0.0846700 & 0.2841897 & 0.7136759 & 0.0062743 & 0.0256386 & 0.1314096 \end{bmatrix} \tag{3.74}$$

after 45 recursions using the simple iteration given in equation 3.59 and a demanded precision of $\|R_{(k)} - R_{(k-1)}\|_\infty < 10^{-6}$. Using this we get the state probability distribution shown in figure 3.26; where also the state probabilities for less loaded situations (*halfe*- and *quarter*-load) are shown, together with the mean metrics table. The figure shows the state probabilities by bold lines, the phase

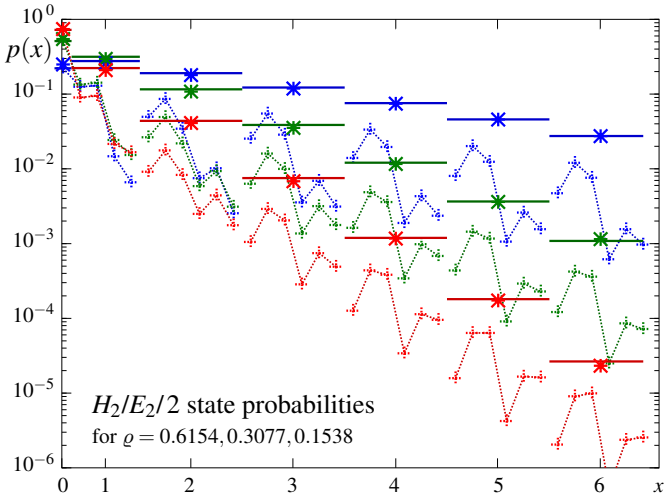


Figure 3.26: State probabilities p_i of the $H_2/E_2/2$ queueing system example

probabilities by dotted '+' symbols, and the simulation result by solid '*' symbols. Note that state and phase probabilities are discrete; for better visibility the phase probabilities that contribute to the same level are connected by a dotted line. The shape of these connecting lines show similarities toward higher states more distant from the boundary level, but also depend on the actual load if we compare them vertically.

To actually calculate the mean metrics shown in table 3.1, we need a fitting equation to correctly calculate $\mathrm{E}[X]$. The common equation 3.62 cannot be applied straight because the boundary level comprises phases where the system state is not zero. Actually equation 3.62 yields the population of the system less the one that is already in the system prior entering and post exiting the repeating levels covered by $\pi_i = \pi_1 R^{i-1} \forall_{i>0}$. This customer is present in all states except the true idle state and thus, we can add it with probability $1-p_{00}$ and get

$$\mathrm{E}[X] = 1\,(1-p_{00}) + \left\|\pi_1 (I-R)^{-2}\right\|_1 \tag{3.75}$$

where p_{00} is the probability of the idle state, here $p_{00} = \pi_0(1)$, the first element of the boundary phase probabilities π_0, which result from equation 3.60 and 3.61.

The performance metrics shown in table 3.1 and figure 3.27 do not match well. While the confidence intervals for the by simulation measured $\mathrm{E}[X]$ shrink for decreasing loads, the calculated

Table 3.1: Calculated *mean performance metrics* of the $H_2/E_2/2$ queueing system example

λ_g	$\mathrm{E}[X]_{\mathrm{calc}}$	$\mathrm{E}[X]_{\mathrm{sim}}$	$\mathrm{E}[T_f]_{\mathrm{calc}}$	$\mathrm{E}[T_f]_{\mathrm{sim}}$	$\mathrm{E}[T_W]_{\mathrm{calc}}$	$\mathrm{E}[T_W]_{\mathrm{sim}}$	$\mathrm{E}[Q]_{\mathrm{calc}}$	$\mathrm{E}[Q]_{\mathrm{sim}}$
0.6154	2.0661	1.9614	3.3574	3.1873	1.3574	1.1873	0.8354	0.7306
0.3077	0.7405	0.6971	2.4067	2.2657	0.4067	0.2657	0.1251	0.0818
0.1538	0.3389	0.3178	2.2027	2.0655	0.2027	0.0655	0.0312	0.0101

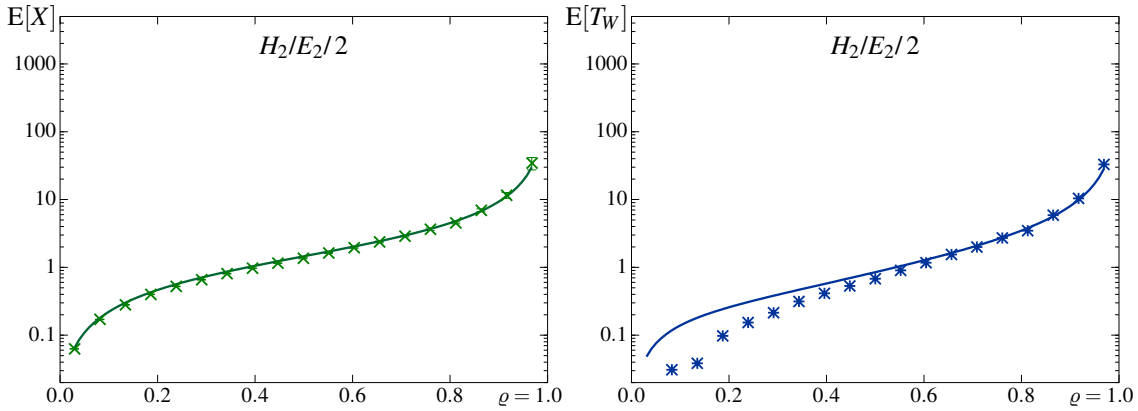


Figure 3.27: Mean system filling $\mathrm{E}[X]$ and waiting time $\mathrm{E}[T_W]$ of the $H_2/E_2/2$ queueing system

results, particularly for the mean waiting time $\mathrm{E}[T_W]$, diverge more and more toward low loads. This questions if the matrix geometric method can be applied to analyse the example $H_2/E_2/2$ queueing system and systems alike where the boundary level comprises a *phase structure* different from that of repeating levels. Actually, the boundary level influences the system performance the more the less the system is loaded. A formal prove is via the mathematics involved is out of the scope, and as well might equation 3.75 be inappropriate. However, as stated in [34] and others, the matrix geometric method is at least applicable for the single server *Ph*/*Ph*/1 case where true QBD structures occur with solely idle phases in the boundary level. Changing the model to a single server system with the same gross capacity we get the results shown in figure 3.28. For the $H_2/E_2/1$ system the calculated and

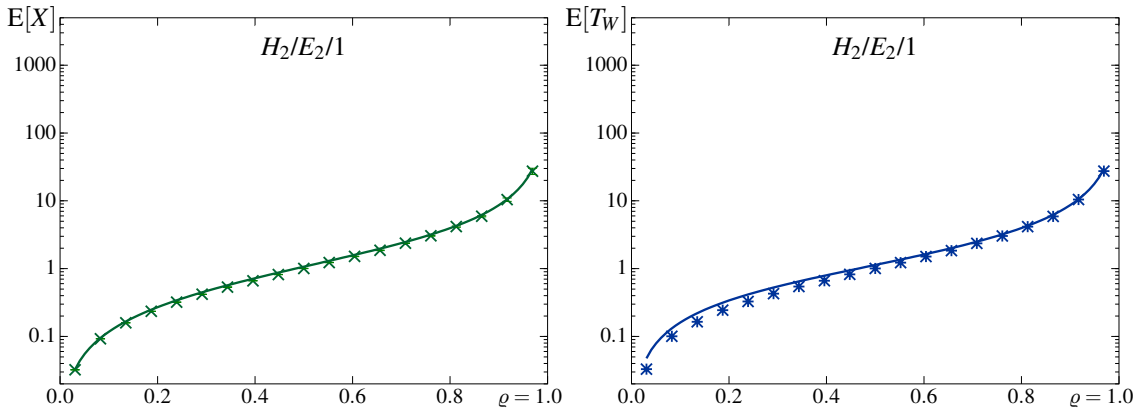


Figure 3.28: Mean system filling $\mathrm{E}[X]$ and waiting time $\mathrm{E}[T_W]$ of the $H_2/E_2/1$ queueing system

measured results fit better, and assuming that the remaining error is due to numeric calculation issues we may agree that the matrix geometric method can be used to analyse single server phase-type queueing systems. However, the reason for the divergence is not identified yet and the fault may reside somewhere hidden in the simulation. In that case the matrix geometric method as outlined above may have yielded the correct results for the $H_2/E_2/2$-system. Still, a proper scientific prove is missing. But how to apply the method in principle should be clear from the provided example.

3.2 Finite queueing systems

In practice all systems are finite if we agree that accessible space is finite. If $p_{ijk}(i{=}s)$ is negligible, infinite queues approximate finite queues. However, as shown in figure 3.29 remain finite systems stable for $\varrho \geq 1$, and yield rising steady state probabilities, $p(i) < p(i+1)$, in overload. The steady state probabilities p_{ijk} of infinite systems need to decrease, at least in average, for stability reasons.

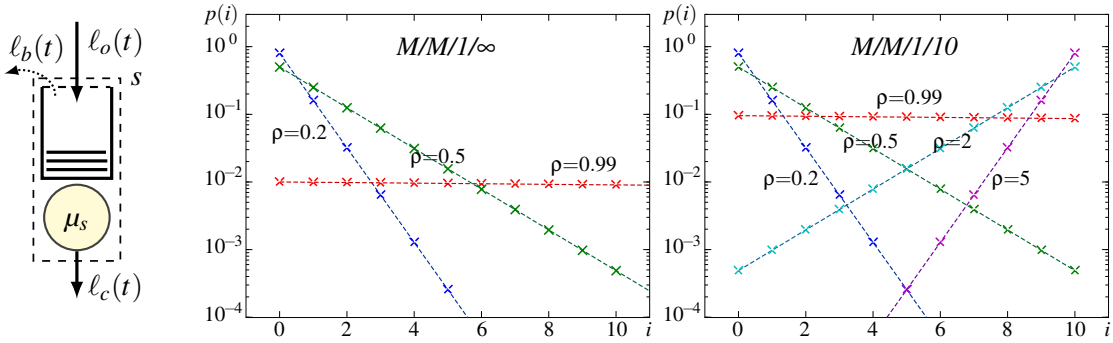


Figure 3.29: Steady state probabilities $p(i)$ for *M/M/1/∞* and *M/M/1/10* at different loads $\rho{=}\frac{\lambda}{\mu}$

With infinite queuing systems the system filling X and the waiting time T_w constitute the main system characteristics, besides the server utilization U, the related queue filling $Q = X - U$, and the flow time $T_f = T_w + T_s$. To characterise finite queueing systems with their bounded number of waiting customers we recognise another primary performance metric, the *blocking probability Pb*

$$P_b = \sum \frac{\lambda - \sum r_{ijk,(i+1)jk}}{\lambda} p_{ijk} \tag{3.76}$$

where we actually sum over states that do not have outbound state transitions that cover all arrival events, precisely states where $\sum r_{ijk,(i+1)jk} < \lambda$. Commonly this occurs when $i{=}s$ and with $\sum r_{ijk,(i+1)jk} = 0$, whereas for all other states $\sum r_{ijk,(i+1)jk} = \lambda$, such that their probabilities are not considered in equation 3.76. P_b expresses the *percentage of time* during which the system may not accommodate new customers. Multiplied with λ it yields the *blocking rate* δ, being the mean number of rejected customers, also known as *overflow* when the process is evaluated.

With blocked arrivals the correct notation of flow volumes (rates) becomes critical. The two terms *offerd load* $\ell_o(t)$ and *carried load* $\ell_c(t)$ identify the load that tries to enter the system and the load that is allowed to pass the system respectively, as shown by the system sketch in figure 3.29. Their difference is called *blocked load* $\ell_b(t) = \ell_o(t) - \ell_c(t)$. In average the carried load equals the departure rate $\vartheta = \mathrm{E}[\ell_c(t)]$, if all clients that are allowed to enter actually become served (no exit without service). The relation *mean carried over offered load* is given by $(1 - P_b)$, such that

$$\vartheta = \mathrm{E}[\ell_c(t)] = (1 - P_b)\,\mathrm{E}[\ell_o(t)] = (1 - P_b)\,\lambda \tag{3.77}$$

and its divergence from the arrival rate defines the *mean blocking rate* $\delta = \mathrm{E}[\ell_b(t)]$:

$$\delta = \mathrm{E}[\ell_b(t)] = \mathrm{E}[\ell_o(t) - \ell_c(t)] = \lambda - \vartheta = \lambda - (1 - P_b)\lambda = P_b\,\lambda \tag{3.78}$$

The load related mean rates, $\lambda, \vartheta, \delta$, can be stated in *Erlang*, being the average number of *arrivals, departures, or losses, per mean holding time* $h{=}\frac{1}{\mu}$, respectively. Commonly, the *Erlang* unit is initially stated with the offered load λ to express that in consequence every time related metric is based on the predefinition $h{=}1$, meaning that the average holding time h defines the applied time unit, such that $\rho{=}\frac{\lambda}{\mu}{=}\lambda$ because $\mu{=}\frac{1}{h}{=}1$. For multiple servers the system capacity is than $\mu_s{=}n\mu{=}n$.

Interchangeably the carried load $\ell_c(t)$ and the mean departure rate ϑ will be termed *throughput*. In some literature the term 'throughput' specifies the maximum carried load, which we call *system capacity* or μ_s (system service rate) for consistency.

State transition diagrams

A state transition diagram depicts all possible system states as nodes together with all possible state changes, represented by the edges in the diagram, which refer to some event that occurs. The edges define the possible change directions, being the neighbour states that can become the next system state when the current state is left due to the occurrence of an arrival, departure, or any other event.

In the time continuous regime every event, arrival or departure, causes a dedicated up or down transition because two events cannot occur at the same infinitesimal small time instant. The rate r_{ij} given with the edges in the state transition diagram is the rate at which transition causing events occur, conditional to the system being in the transition's originating state. To express the nature of this parameter the diagram is sometimes called *state transition rate diagram* to separate it from the *state transition probability diagram* common with time discrete systems. Anyhow, if transition options exist for an event, their selection probabilities, here α_{ij}, β_{ji}, need to sum up to one per state and event, $\sum_j \alpha_{ij} = 1$, $\forall_i$ and $\sum_i \beta_{ji} = 1$, $\forall_j$. The *state transition probabilities* $p_{ij} = \mathrm{P}[i \to j]$ over

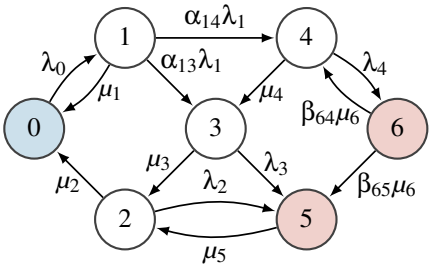


Figure 3.30: Arbitrary state transition *rate* diagram

time result from multiplying the state probability p_i of the source node at the origin of a transition with the *conditional event occurrence rate* $(\lambda_i, \mu_i, \varepsilon_i, \ldots)$ stated in the diagram next to edges, possibly decreased by some *splitting factors* $(\alpha_{ij}, \beta_{ji}, \gamma_{ij}, \ldots)$ where with certain proportionality different state changes occur in response to an event. The state probabilities p_i, being the probabilities that the system is in state i over time, result from solving the equation system that reflects the state transition diagram mathematically. Evidently, the system needs to be in some state at all times, and thus, $\sum p_i = 1$ is a mandatory condition that has to be fulfilled at any time.

To calculate the state probabilities p_i for the steady state we need to solve:

$$\text{the linear equation system} \qquad Q\,p = 0 \tag{3.79}$$

$$\text{constrained to} \qquad \sum_i p_i = 1 \tag{3.80}$$

This applies for any state transition diagram, however complex it my be, and for finite systems this yields a finite set of linear equations that always can be solved. The methods proposed to perform this more efficiently are sometimes restricted to special topologies, for example diagonal Q matrices as they occur with QBD systems.

To perform transient analysis in order to evaluate the response of the system to an environmental change, we need to solve the differential equation system, i.e.:

$$\text{the differential matrix equation} \qquad \dot{p}(t) = Q\,p(t) \tag{3.81}$$

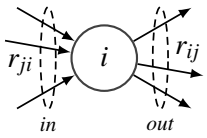
$$\text{constrained to} \qquad \sum_i p_i(t) = 1 \;\forall_t \tag{3.82}$$

In the context of finite queueing systems this has been intensely studied by Markus Sommereder in [76]. For the assessment of a queueing system's performance are transient effects negligible only if they decay in a sufficiently short time without causing problems, such that the system operates in the steady state most of the time.

Steady state analysis

Steady state does not mean that the system is in single state all the time. What actually does become steady is the probability that the system is in a certain state, and therefore the average proportion of time that the system spends in each state while the system continues to change its state.

Stable probabilities can only occur when the total transition probability at which the system enters a certain state equals the total transition probability at which it leaves this state. This is known as the *balance condition*, and mathematically expressed by the so called *equilibrium equations*.



$$\sum_j p_j r_{ji} = \sum_j p_i r_{ij} \; \forall_i \tag{3.83}$$

If we specify equation 3.83 $\forall i$, we get i equations for the i unknown state probabilities p_i. However, these are linearly dependent, so we need one more equation to solve the equation system. Actually, the condition stated in equation 3.80 yields this additional equation required to calculate all state probabilities p_i. It can be integrated in the calculation as outlined in section 1.4.3.

Note that equilibrium equations can be stated likewise for state groups if the grouping separates the entire state space into two groups that contain in total all states. If one group contains a single state only, we get the equations state in equation 3.83. Complexity and calculation effort can be effectively saved by using *macro*-states to jointly represent state groups whenever this is applicable.

Commonly, stable dynamic systems propagate toward a steady distribution of state probabilities as long as the external conditions (environmental parameters) do not change. Exceptions are multivalent and chaotic/fractal systems, which may show strange and possibly misleading system behaviour. If measured performance metrics of a stable system do not stabilise with extended observation time, than this is likely due to unstable environmental conditions. Only if latter can be reliably excluded, we may conclude that the system is multivalent or chaotic.

Approximation by truncation

The challenge of finite state transition systems is the potential size of the equation system and the transition matrix in particular. If low load analysis $\varrho < 1$ is sufficient and an analytic result for the same but infinite queueing system is available, than the *approximation by truncation* can be used to achieve approximate results, as sketched in figure 3.31.

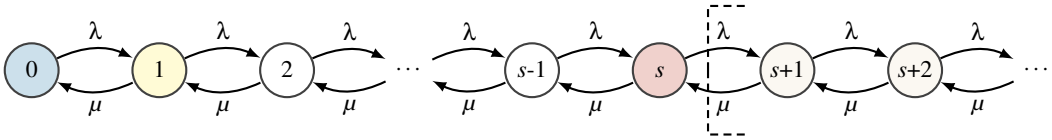


Figure 3.31: Truncation of *M/M/1* state transition diagram to get *M/M/1/s*

Skipping the states for $i > s$ without adjusting the state probabilities p_i^* calculated via the equation for the infinite *M/M/1* system, $p_i^* = (1-\rho)\,\rho^i$, would violate $\sum p_i = 1$. Therefore, the remaining state probabilities $p_i^* \; \forall_{i \le s}$ need to be increased such that the condition is fulfilled. This is achieved by re-normalisation:

$$p_i = \frac{p_i^*}{\sum_{k=0}^{s} p_k^*} = \frac{p_i^*}{(1-\rho)\sum_{k=0}^{s} \rho^k} = \frac{p_i^*}{1-\rho^{s+1}} = \frac{1-\rho}{1-\rho^{s+1}}\,\rho^i \tag{3.84}$$

For *M/M/1/s* this approach yields the precise result. Generally speaking the negative exponential distribution approximates any unknown distribution, and thus, the *truncation and re-normalisation* approach can be applied generally to achieve approximate results at least.

3.2.1 *M/M/n/s* queueing systems

To get the *M/M/n/s* system's state transition diagram we only need to drop all states with $i > s$ from the infinite system's state transition diagram shown in figure 3.2 in order to consider the size bound s. The resulting state transition diagram is shown in figure 3.32. As before, for $i \geq n$ all servers are

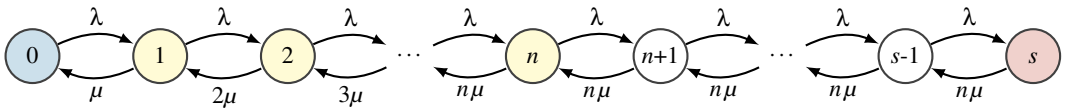


Figure 3.32: *M/M/n/s* state transition diagram

busy and the current service rate μ_i equals the system capacity μ_s, $\mu_i = n\mu \; \forall_{i \geq n}$, whereas for $i < n$ it relates state dependently to the number of currently busy servers, $\mu_i = i\mu < \mu_s \; \forall_{i<n}$.

For finite queueing system the state space does not extend infinitely to the right. It is bounded by the *blocking state*, $i = s$, and while the system remains in this state s, all customers that may arrive become *blocked*, such that they do not enter the system and thus cause no state transition.

The state transition diagram is finite, thus we can set up a finite *system of equilibrium equations*

$$\begin{aligned} p_0\lambda &= p_1\mu && i = 0 \\ p_i(\lambda + i\mu) &= p_{i-1}\lambda + p_{i+1}(i+1)\mu && 0 < i < n \\ p_i(\lambda + n\mu) &= p_{i-1}\lambda + p_{i+1}n\mu && n \leq i < s \\ p_s n\mu &= p_{s-1}\lambda && i = s \end{aligned}$$

that again can be rewritten in matrix form as $Q\,p = 0$ and solved by including the condition $\sum p_i = 1$ as outlined in section 1.4.3. The transition matrix has the tri-diagonal QBD form

$$Q = \begin{bmatrix} -\lambda & \lambda & & & & & & & \\ \mu & -(\lambda+\mu) & \lambda & & & & 0 & & \\ & 2\mu & -(\lambda+2\mu) & \lambda & & & & & \\ & & \ddots & \ddots & \ddots & & & & \\ & & & n\mu & -(\lambda+n\mu) & \lambda & & & \\ & & & & n\mu & -(\lambda+n\mu) & \lambda & & \\ & & & & & \ddots & \ddots & \ddots & \\ & 0 & & & & & n\mu & -(\lambda+n\mu) & \lambda \\ & & & & & & & n\mu & -n\mu \end{bmatrix}$$

and as required are all row-sums *zero*. Alternatively, we can rewrite the equilibrium equations into a recursive calculation procedure

$$\begin{aligned} p_1 &= \tfrac{\lambda}{\mu} p_0 && i = 0 \\ p_{i+1} &= \tfrac{i}{i+1} p_i - \tfrac{\lambda}{(i+1)\mu}(p_{i-1} - p_i) && 0 < i < n \\ p_{i+1} &= p_i - \tfrac{\lambda}{n\mu}(p_{i-1} - p_i) && n \leq i < s \\ p_s &= \tfrac{\lambda}{n\mu} p_{s-1} && i+1 = s \end{aligned}$$

that allows to relate all state probabilities p_i^* to an assumed one p_a^*. Using the condition $\sum p_i = 1$ we finally find the correction term required to get the absolute state probabilities

$$p_i = \frac{p_i^*\, p_a^*}{\sum p_i^*}$$

where we commonly assume $p_a^*{=}p_0^*{=}1$ for calculation simplicity.

With recursive equation insertion the known analytic equations for all p_i can be derived from the recursive equilibrium equations. This yields for example the geometrically distributed state probabilities $p_i = \rho^i(\frac{1-\rho}{1-\rho^{s+1}})$ for the *M/M/1/s* queueing system with $\rho = \varrho = \frac{\lambda}{\mu}$ shown in figure 3.29. The recursive approach is straightforward for one-dimensional state transition diagrams. In principle it works for any state transition diagram, but may require more assumed state probabilities and accordingly more corrective relations, which can become cumbersome. Commonly, we use the *matrix form* $\underline{Q}\,\vec{p} = \vec{0}$ and let a computer solve the equation system for some given λ, μ, n, and s.

If mean values are required to assess a given system's performance only, we can apply straight calculation from state probabilities,

$$\mathrm{E}[X] = \sum_{i=0}^{s} i\, p_{ijk} \tag{3.85}$$

$$\mathrm{E}[Q] = \sum_{i=n+1}^{s} (i-n)\, p_{ijk} = \sum_{i=n+1}^{s} j\, p_{ijk} \tag{3.86}$$

$$\mathrm{E}[U_{abs}] = n\,\mathrm{E}[U_{rel}] = \sum_{i=0}^{n} i\, p_{ijk} + \sum_{i=n+1}^{s} n\, p_{ijk} = \mathrm{E}[X] - \mathrm{E}[Q] \tag{3.87}$$

$$P_b = \sum_{i=s} p_{ijk} \quad \rightarrow \quad \ell_b = P_b\,\lambda \quad \rightarrow \quad \ell_c = (1-P_b)\,\lambda \tag{3.88}$$

$$\mathrm{E}[T_w] = \frac{\mathrm{E}[Q]}{\ell_c} = \frac{\mathrm{E}[Q]}{(1-P_b)\,\lambda} \tag{3.89}$$

$$\mathrm{E}[T_f] = \frac{\mathrm{E}[X]}{\ell_c} = \frac{\mathrm{E}[Q]+\mathrm{E}[U]}{(1-P_b)\,\lambda} = \mathrm{E}[T_w] + \frac{\mathrm{E}[U]}{(1-P_b)\,\lambda} = \mathrm{E}[T_w] + \tfrac{1}{\mu} \tag{3.90}$$

$$\mathrm{E}[U_{rel}] = (1-P_b)\,\tfrac{\lambda}{n\mu} = (1-P_b)\,\varrho \tag{3.91}$$

because with finite systems it is actually possible to calculate all state probabilities and thereby the system metrics, a numerical impossibility with infinite systems.

♢ Note that for Little's law to be correct we have to use the *carried load* $\ell_c = (1-P_b)\,\lambda$ to specify the relevant *ingress rate* 'λ', and not the offered load $\ell_o = \lambda$. A peculiarity indiscernible with infinite systems where $\ell_c = \ell_o = \lambda$. Correctly, we should write $N = \lambda_{in} T$ when stating Little's law.

♢ The server utilization $\mathrm{E}[U]$ implicitly states the *throughput* ϑ:

$$\vartheta_{M/M/n/s} = n\mu\,\mathrm{E}[U_{rel}] = \mu\,\mathrm{E}[U_{abs}] = (1-P_b)\,\lambda \tag{3.92}$$

If the system utilization is given relative, meaning $\mathrm{E}[U] \in [0,1]$ as for example in figure 3.33 and 3.34, it implicitly depicts the *throughput* ϑ relative to the *system capacity* $n\mu$. The absolute $\mathrm{E}[U] \in [0,n]$, as it for example results from equation 3.87, states the *throughput* ϑ in relation to the *service time* $h = \frac{1}{\mu}$, as implied with the *Erlang* unit definition. For $n = 1$ the two $\mathrm{E}[U]$ metric variants coincide.

♢ Of particular relevance is equation 3.91 because it expresses analytically the relation among *load* ϱ, *blocking* P_b, and *server utilization*. Evidently causes low load poor utilization, but also high blocking probability limits the achievable server utilization.

To illustrate how the performance depends on the system size s, figure 3.33 depicts the mean system filling $\mathrm{E}[X]$, the mean waiting time $\mathrm{E}[T_w]$, the mean server utilisation $\mathrm{E}[U]$, and the blocking probability P_b over increasing system load ϱ of *M/M/n/s* queueing systems with three servers, $n = 3$, and increasing size $s = [3, 9, 30, 90]$. The mean system filling $\mathrm{E}[X]$ and waiting time $\mathrm{E}[T_w]$ saturate according to the system size limit. The more space is provided by the queueing system, the more load can be buffered and may thus wait for being served. This heavily decreases the blocking

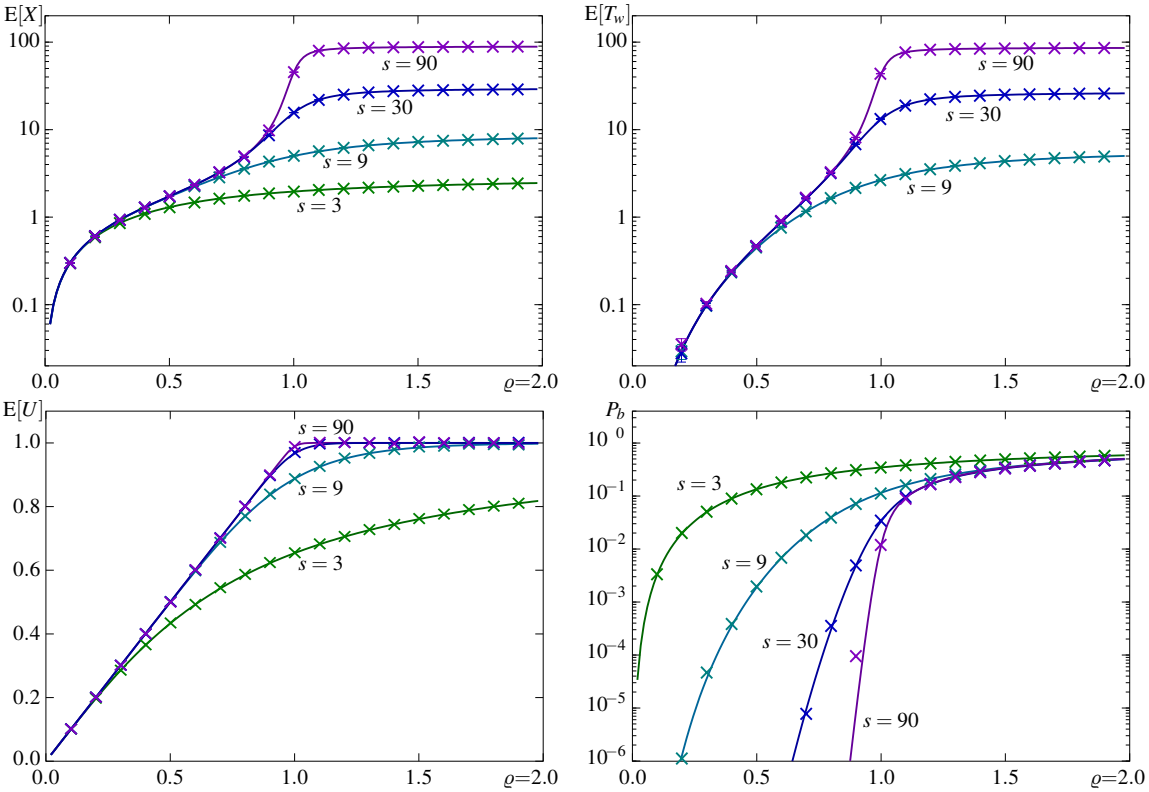


Figure 3.33: *M/M/3/s* queueing system performance (calculated *curves* and simulation results ×) over the system load ϱ for different system size $s = [3,9,30,90]$ and $n\mu = 1$, showing the mean system filling $\mathrm{E}[X]$, waiting time $\mathrm{E}[T_w]$, server utilisation $\mathrm{E}[U]$, and blocking probability P_b

probability P_b, and the traces for the server utilization $\mathrm{E}[U]$ show that with Markovian arrivals a system size ten times the system capacity, $s = 10n\mu$, approximates the optimum quite well. We also recognise that only for very small system sizes $s < 3n\mu$, meaning a queue size less than twice the system capacity, degrades the system utilization notably outside a rather narrow range around $\varrho = 1$. For the special case $s = n$ no queue exists, and therefore, no loads can wait for service, such that $T_w = 0\,\forall_\varrho$. In this case, all loads that cannot be served immediately become blocked, and therefore, these systems are commonly called *loss systems*. Such systems will be discussed more detailed in section 3.3.1, along with other queue-less systems of practical relevance.

At mean loads common with telecommunication infrastructures $\varrho < 0.3$ we notice that P_b is far below 10^{-6} for $s \geq 10n$, and all other performance metrics shown in figure 3.33 are also very close to those found for the corresponding infinite systems with $s \to \infty$.

To study the influence of the number of servers, figure 3.34 shows the performance of *M/M/n/s* systems over the number of provided servers n at high load $\varrho = 0.8$ and equal system capacity $n\mu = 1$. To compare similarly *scaled M/M/n/s* systems with $n \leq s$, the system size s is set to be a multiple of the server count n, particularly $s/n = 1,3,10,30$. Again, for $s = n$ no waiting space exists, and thus, no curve for the waiting time in case of $s/n = 1$ exists. The system filling $\mathrm{E}[X]$ rises with the number of servers n because the mean service time $h = \frac{1}{\mu}$ must increase for constant system capacity $n\mu$. The mean waiting time $\mathrm{E}[T_w]$ is influenced by two factors: on one side the number of servers n that causes decreasing T_w for increased n, and on the other side by the finiteness of the queue, $s_q = s - n$, which upper binds T_w. Because of defining the queue size s as multiple of the server count n we get partially increasing mean waiting times $\mathrm{E}[T_w]$ over n when the increased carried load $\lambda(1-P_b)$ outperforms the advantage gained from more servers. At overload, $\varrho > 1$, this factor dominates and for all s/n-ratios T_w increases with n. However, if we would keep s constant and the system

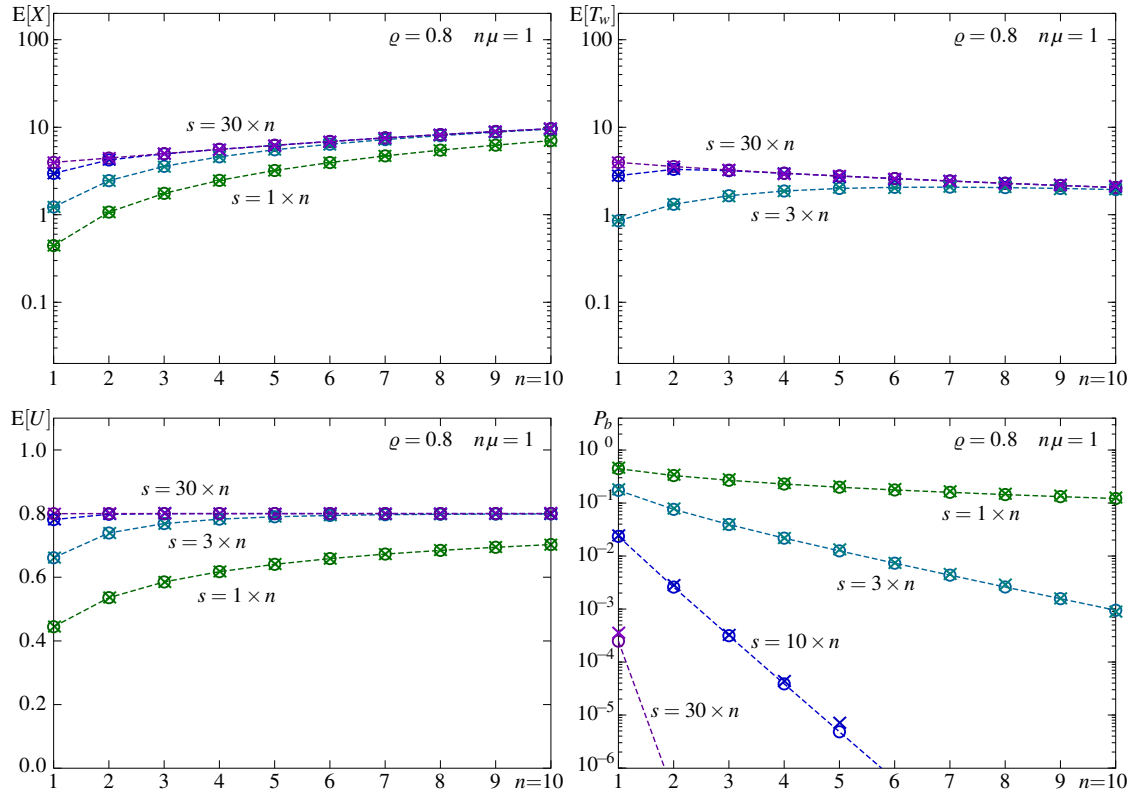


Figure 3.34: *M/M/n/s* queueing system performance (○ - calculated, × - simulated results) over the number of servers n for related system size, $s/n = [1,3,10,30]$, at high load $\varrho = 0.8$ and $n\mu = 1$

load below overload, $\varrho < 1$, we get decreasing T_w for increasing n, as found with the corresponding infinite systems presented in section 3.1.1 figure 3.5.

In accordance with the conclusions drawn from figure 3.33, we again recognise that a system size 10-times the system capacity $\mu_s = n\mu$ performs quite close to optimum, at least for Markovian arrival and service processes. Form figure 3.34 we can conclude in addition that the load splitting across more but accordingly less powerful servers has a rather low influence on the system performance. The strong reduction of the blocking probability P_b is gained from the implicit queue size increase, $s_q = s - n = (s/n - 1) \times n$, and not from the increased work-load splitting among more servers. The shown $\mathrm{E}[X]$ over n indicates that without the queue size scaling the blocking probability P_b should slightly rise with increasing n when keeping the system capacity $\mu_s{=}n\mu$ constant.

For completeness we briefly cite the analytic results for *M/M/1/s*, which likely are to be found in every textbook on queueing systems, and refer to Kleinrock's chapter 3.6 on finite storage in [14].

$$p(0) = \frac{1-\rho}{1-\rho^{s+1}} \quad \rightarrow \quad p(i) = \begin{cases} p(0)\rho^i, & \rho \neq 1 \\ \frac{1}{s+1}, & \rho = 1 \end{cases} \quad \rightarrow \quad P_b = \begin{cases} p(0)\rho^s, & \rho \neq 1 \\ \frac{1}{s+1}, & \rho = 1 \end{cases}$$

$$\mathrm{E}[X] = \begin{cases} \frac{\rho}{1-\rho} - \frac{(s+1)\rho^{s+1}}{1-\rho^{s+1}}, & \rho \neq 1 \\ \frac{s}{2}, & \rho = 1 \end{cases} \quad \rightarrow \quad \mathrm{E}[T_f] = \frac{\mathrm{E}[X]}{(1-P_b)\lambda} \quad \rightarrow \quad \mathrm{E}[T_w] = \mathrm{E}[T_f] - \frac{1}{\mu}$$

Calculating the queue filling $\mathrm{E}[Q]$ from the waiting time $\mathrm{E}[T_w]$ we recognise the factor $(1-P_b)\,\rho$, which is the *server utilization* $\mathrm{E}[U] = \mathrm{E}[X] - \mathrm{E}[Q]$ that states the system *throughput* $\vartheta = \mathrm{E}[U]\,\mu$.

$$\mathrm{E}[Q] = (1-P_b)\,\lambda\,\mathrm{E}[T_w] = \mathrm{E}[X] - (1-P_b)\,\rho \quad \rightarrow \quad \vartheta = (\mathrm{E}[X] - \mathrm{E}[Q])\,\mu = (1-P_b)\,\lambda$$

3.2.2 Queueing disciplines

The common discipline is *first in first out* (FIFO) queueing, which corresponds to pipelined operation. The opposite is *last in first out* (LIFO) queueing, which corresponds to stacked operation. Another special case sketched in figure 3.35 is *random queueing* (RAND). The mean system performance

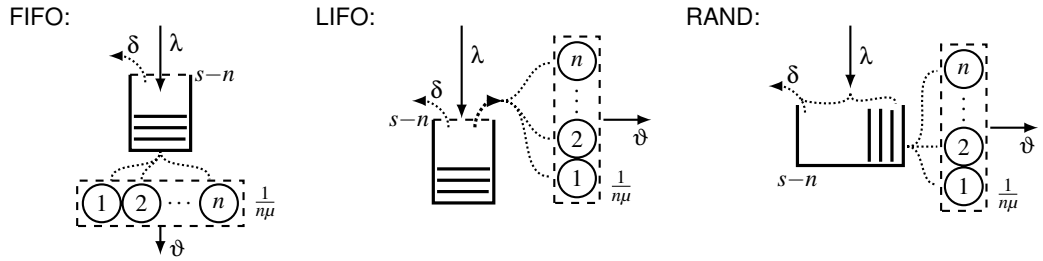


Figure 3.35: Different queueing disciplines: FIFO, LIFO, and random queueing

metrics stated by equation 3.85 to 3.91 do not depend on the queueing discipline because the state probabilities p_i themselves do not depend on the applied discipline. Therefore, these metrics can be calculated directly from the state probabilities p_i, as given by the above stated equations. However, the *variation of the waiting time* $Var(T_w)$, constituting the forwarding *jitter*, does depend on the discipline. Intuitively causes LIFO queueing most jitter, whereas least jitter results from FIFO queueing. Therefore, these two disciplines are required to assess the upper and lower bounds in case the actual discipline cannot be modelled. Somewhere in between resides *random queueing* (RAND), which may be of interest to state an expectable jitter for system implementing complex, possibly flow aware queueing disciplines that cannot be modelled in detail.

The waiting process

To evaluate the *waiting process* T_w we use the so called *state flow diagram*. It results from the state-transition-diagram when we assume that the system is with probability p_i in some state when a *test*-client arrives, and depict all possible paths via intermediate states feasible to reach some absorbing state that terminates the waiting process. Obviously, there may exist many paths to transit from one state to an absorbing state, particularly if forward-backward loops are possible as for example with LIFO queueing. However, solving the system of differential equations for all possible arrival states yields results that consider all path alternatives to any absorbing state without the need to explicitly analyse every possible path variant. The waiting process T_w is than the sum of all the different distributions found from solving the state flow diagram, each weighted by p_i, the probability for an arrival to occur in state i, which we know from solving $Qp=0$.

Clients that arrive while at least one server is idle, when $i<n$, do not wait at all. These are served immediately, and add a Dirac impulse with intensity $p_{i<n}$ at $t_w=0$ to the waiting time distribution, independent of the queueing discipline applied. And in any case, we cannot consider the theoretically infinite waiting time of blocked clients.

The state flow diagram for FIFO queuing, shown in figure 3.36, is rather simple. For the waiting

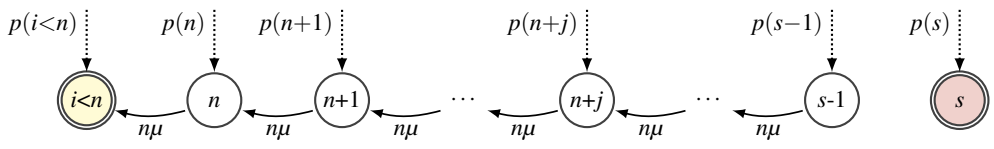


Figure 3.36: *M/M/n/s*/FIFO state-flow-diagram of the waiting time process T_w

time evaluation are all states $i<n$ irrelevant and therefore can be summarised in a single macro state,

such that only the homogenouse queueing part with service rate $n\mu$ remains. Once the test client entered the system with i clients in front, which occurs with probability $p(i)$, it moves forward to being served with every service completion of a client in front of it, at rate $n\mu$, because later arrivals have no influence on it.

This changes completely for non-preemptive LIFO queueing, shown in figure 3.37, where the

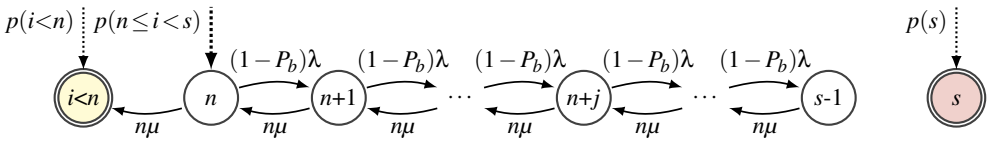


Figure 3.37: *M/M/n/s/*LIFO state-flow-diagram of the waiting time process T_w

test client enters the queue at the queue's head, with probability $p(n \le i < s)$, if accepted and not served immediately. Once queued, it becomes pushed back whenever another client is accepted.

For finite queues we need to state a strategy that defines which client is dropped if an arrival occurs when the queue is full. For consistency with the other queueing disciplines and in accordance with no pre-emption, we assume that no arriving client may push out an already queued client. This implies that we need to reduce the arrival rate (load) to the carried load $(1-P_b)\,\lambda$ (throughput ϑ), which covers accepted clients only.

The state flow diagram for RAND queueing shown in figure 3.38 is a little more complex than the extremes presented above. Here all clients that happen to be queued in front of the test client, independent when these arrived, need to be served prior the test client can enter a server. It is possible to represent this by a state flow diagram based on those depicted above for FIFO and LIFO, but it is easier if we assume random serving, meaning that a client is selected randomly from all currently waiting in the queue, such that the queuing order is irrelevant. However, this causes that the state index in figure 3.38 no more indicates the number of clients in front of the test client. The index now states the total number of *competing* clients currently in the system, not including the test client.

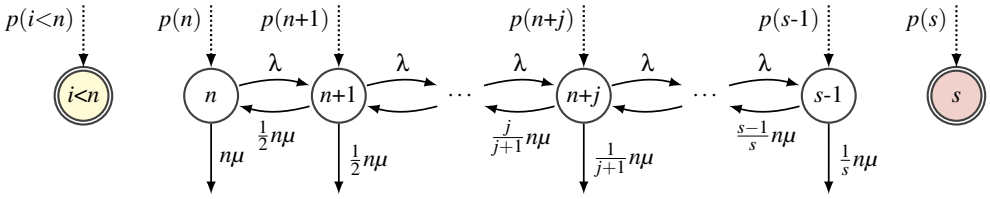


Figure 3.38: *M/M/n/s/*RAND state-flow-diagram of the waiting time process T_w

With probability $p(n+j)$ the test client enters the system with j waiting clients, although at an arbitrary position in the queue. With likelihood $\frac{1}{j+1}$ the test client is the next to be served. This likelihood remains $\frac{1}{j+1}$ when later arriving clients increase the number of competing clients j, which occurs with rate λ, as long as the queue is not full. As before, clients that arrive when the queue is full are blocked, such that no client once queued becomes pushed out. As the state indices do not including the test client itself, flow state s cannot be reached while the test client is in the system.

The flow process

The flow process is the waiting process plus the service time of the client itself. Here we need to consider the non-zero contribution from the clients that are served immediately on arrival. To get the state flow diagram of the flow time we need to add the test client's service phase, as shown for example in figure 3.39 for random queueing. To be a memoryless process, the future cannot depend on how the current state has been reached. Actually, for non-preemptive serving we may

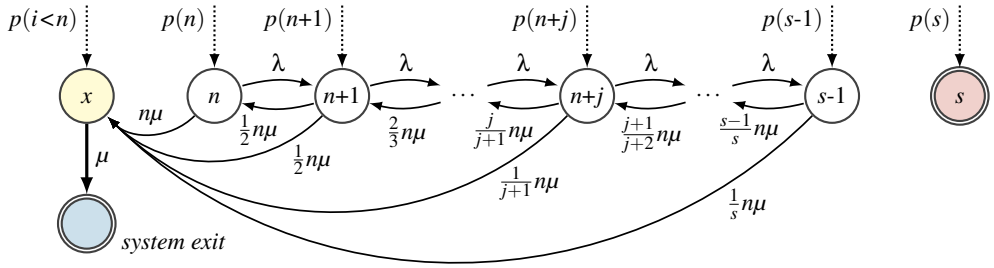


Figure 3.39: State-flow-diagram of T_f for a non-preemptive *M/M/n/s/*RAND queueing system

forget all competing clients once the test client is served. Thus, we indicate the serving state by an x because we do not remember how many clients were left behind. As before, the resultant system of differential equations can be solved to get the flow time distribution $T_f(t)$.

Note: if an author prefers to identify the states with the number of clients in the system including the observed test client, than all transitions need to be shifted to the right by one state. Note that in that cases the weighting of the flow- and waiting-paths that start in state i is $p_{i\text{-}1}$.

Pre-emptive serving

In case a client may be pre-empted from being served, although for random queuing this is a rather academic assumption, we need to ad transitions back from the serving to a queueing state, which in total occur with rate λ_p, the fraction of privileged arrivals that cause pre-emption. The resultant state flow diagram is shown in figure 3.40, where the dashed transitions to the *blocking* state s exist only if we also skip the policy that no arrival pushes out an already accepted client. In figure 3.40 it is

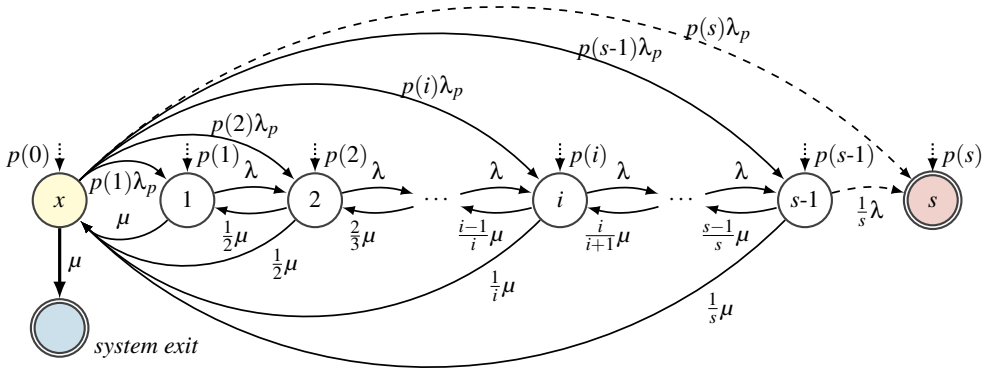


Figure 3.40: MMnsRANDpFlow: State-flow-diagram of the flow time process T_f for an *M/M/n/s/*RAND queueing system with server pre-emption

actually shown for the single server case in order to present the faint differences that occur when we calculate the selection factors using i, the number in the system, instead of the number in the queue j, where $i = j+1$ for $n=1$ server.

With this model we forget how many competing clients remain in the system once the test client is selected for serving. Thus, the pre-emption related transitions are weighted with the probability that there are i competing clients in the system at any time. This works as long as the system is Markovian, meaning that the transition to a next state may depend on the current state only. According to that,

where the pre-empted test client returns to does not need to be the state it came from. Obviously, this does not model reality one-by-one but should not have any influence on the results.

To draw a pre-emption state flow diagram for multiple servers we go back to the state flow diagram shown in figure 3.38 for the waiting process of the *M/M/n/s/RAND* system, and extend that by the individual serving phases and the adjacent transitions including the pre-emption option, as shown in figure 3.41. We get a state flow diagram that explicitly considers all possible paths to

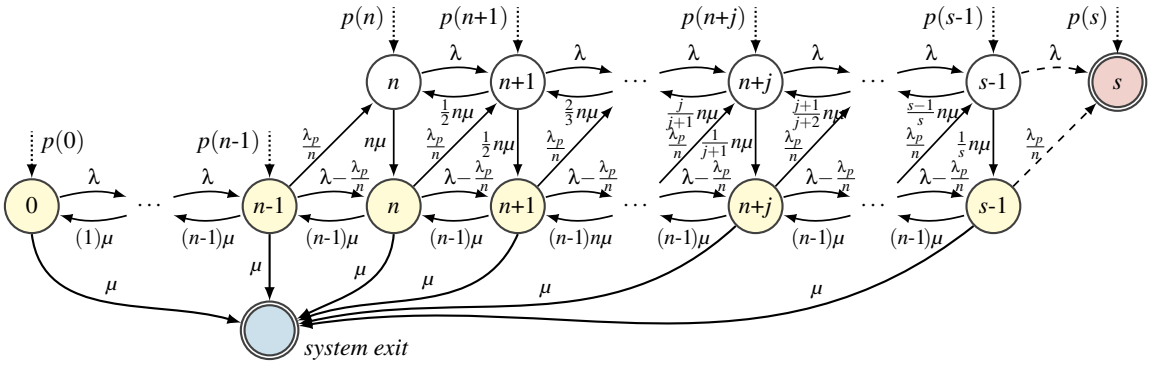


Figure 3.41: MMnsRANDpeFflow: Detailed state-flow-diagram of the flow time process T_f for an *M/M/n/s/*RAND queueing system with pre-emption

the actual state a test client becomes pre-empted to. In addition, we need to consider the applied pre-emption policy, meaning which out of the n clients currently served becomes pre-empted. This reduces the pre-emption rate λ_p by the likelihood for selecting the test client. In case of random selection it becomes $\frac{\lambda_p}{n}$, as assumed here.

We recognise that representing the state flow process can become quite complex, even for purely Markovian systems with a single queue and a single traffic flow. If process phases are introduced, as discussed next in section 3.2.3, or multiple queues and different service time distributions need to be considered for traffic from different flows, the definition of the state flow diagram can hardly be sketched, simply because of the many dimensions required. Still, if we can identify regions of states with similar incoming and outgoing transitions, we can define equilibrium equations that cover an entire region each. These can be sketched one-by-one, depicting a single state with all incoming and outgoing transitions per region only.

Numerical solving $\dot{\vec{p}}(t) = \underline{Q}\,\vec{p}(t)$

To calculate the waiting and flow time distributions presented above we need to solve the system of linear equations stated in equation 3.81. This can be done numerically by applying some capable algorithm, for example the *Runge-Kutta* methods [77]. Working with *Octave* we use the `lsode` procedure, being the "Livermore Solver for Ordinary Differential Equations" [78], specified in the GNU Octave manual as "reliable ODE routines written in Fortran" [29].

This procedure requires a defined function that calculates the right side of the matrix differential equation. For the *M/M/n/s/LIFO* waiting time, and its state flow diagram shown in figure 3.37, this function can for example be defined as outlined here. The function parameters handed over to the *Qp*-function are the initial state probabilities p_0, and the array of times t for which we intend to get results (curve points). For the initial flow state probabilities p_0 we use the *entry probabilities* given by the prior calculated *state probabilities* $p(i)$, as shown in the *state flow diagram* in figure 3.37. The times t are typically a linear array that can be defined by $t = \texttt{linspace}(0, t_{max}, n_{steps})$.

Next we construct row-by-row the Q_f-matrix representing the *state flow process*, starting with the *absorbing state* that has no outbound transitions and thus is represented by a *zero*s row. The

Qp-function for *M/M/n/s*/LIFO waiting time T_w

```
1: function ṗ = Qp(p0,t);                          ▷ parametrised as required for lsode procedure
2: global n, s, ra, rs, p;                          ▷ variables defined globally outside the function
3: re = ra(1−p(i=s));                               ▷ calculate the entry rate (carried load)
4: Qf = [ ]; sQ = s−n+1;                            ▷ initialise empty Qf-matrix, and calculate its final size
5: rowi = zeros(1,sQ);
6: Qf = [Qf ; rowi];                                ▷ add flow absorbing state (the system exit)
7: for i = 2 : sQ−1 do
8:     rowi = zeros(1,sQ);
9:     rowi(i−1) = n rs; rowi(i) = −re − n rs; rowi(i+1) = re;
10:    Qf = [Qf ; rowi];                            ▷ add repeating flow states
11: end for
12: rowi = zeros(1,sQ);
13: rowi(sQ−1) = n rs; rowi(sQ) = −n rs;
14: Qf = [Qf ; rowi];                               ▷ add boundary flow state
15: ṗ = Qf^T p0;                                    ▷ differential equation system to solve
16: endfunction
```

following rows add one-by-one every state to Q_f, left-to-right in figure 3.37. The blocking state $i{=}s$ is not added because it does not contribute to the waiting time distribution. Note that the Q_f-matrix cannot be handed over as function parameter, `lsode` cannot handle any variables other than the stated function parameters. Hence, the Q_f-matrix can be defined outside the Qp-function only if it is defined *global*. Here we kept it inside for clarity, although we use global variables for the number of servers n, the system size s, the arrival rate r_a, the service rate r_s per server, and the steady state probabilities p, to achieve at least some flexibility.

To solve the system of differential equations defined by the Qp-function we execute `lsode` and subsequently calculate and plot the waiting time distribution $F_{T_w}(t)$ together with the components that lead to it. First we create the times array t and set the *initial flow state probabilities* p_0 to the

solve $\dot{\vec{p}}(t) = \underline{Q}\,\vec{p}(t)$ as defined by the Qp-function

```
1: t = linspace(0,tmax,nsteps);                     ▷ time-steps for resultant curves
2: p0 = zeros(s−n+1,1);
3: p0(1) = p(i<n); p0(2) = p(n≤i<s);                ▷ set initial flow state probabilities
4: ṗ = lsode("Qp", p0, t);                          ▷ execute lsode procedure
5: ṗ = ṗ(:,2:end);                                  ▷ remove absorbing state's ṗ
6: ccdf = sum(ṗ,2);                                 ▷ sum-up the conditional components
7: cdf = 1 − ccdf/(1−p(s));                         ▷ calculate the unconditional cdf F_Tw(t)
8: plot(t,[ccdf/(1−p(i<n)−p(s)), ṗ, cdf]);          ▷ plot F^c_{w>0}(t), its components φi(t), and final F_Tw(t)
```

entry probabilities, as defined in the *state flow diagram*. Next is `lsode` executed, where the first parameter is a *text string* that names the *function* to be solved, here the Qp-function's name. When `lsode` finished, we find in the $\dot{p}$-matrix for each flow state i the component $\varphi_i(t)$ that it contributes. The first column is to be ignored, it refers to the absorbing state. The remaining are the components that summed-up and such rescaled that the non-contributing arrivals, those that do not wait and those blocked, are taken out, yield the conditional complementary cumulative distribution function, being the *cccdf* $F^c_{w>0}(t)$ with $F^c_{w>0}(0){=}1$. The *cdf* $F_{T_w}(t)$ results from $1-F^c_{T_w}(t)$, where latter results from scaling, $F^c_{T_w}(t) = F^c_{w>0}(t)\,(1{-}p(s))$, here only considering the blocked part because else the step at $t{=}0$ contributed by the non-waiting clients would not appear.

The results of the shown routines are depicted in figure 3.42. We notice that due to the sole queue entry state in case of LIFO, only its *cccdf* component φ_2 starts with non-zero probability at $t{=}0$. The others are no entry states and therefore their *cccdf* contributions are zero at $t{=}0$. The less likely a flow state is, the smaller is the area beneath the *cccdf* component it contributes, and the more clients

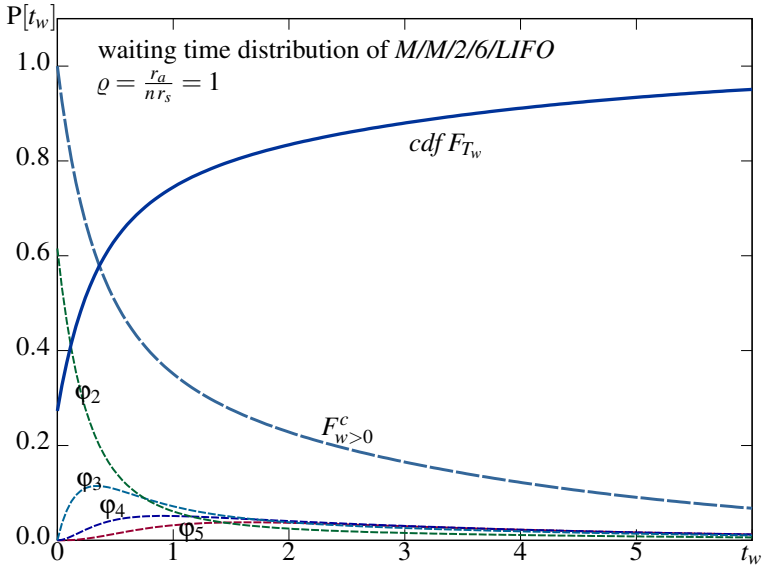


Figure 3.42: Waiting time component's *cccdf*s $\varphi_i(t)$, their normalised sum, the *cccdf* $F^c_{w>0}(t)$, and the resultant *cdf* $F_{T_w}(t)$, as calculated and plotted by the sketched routines for the *M/M/2/6/*LIFO example at load r_a=1, r_s=$\frac{1}{2}$, n=2 → ϱ = 1.

in front of the test client a flow state represents, the further right occurs the peak of the according *cccdf* component. The shown total *cccdf* $F^c_{w>0}(t)$ is more than the sum of the components because of the scaling applied. Finally, the resultant *cdf* $F_{T_w}(t)$ shows a step at t=0; meaning it does not start from zero. This expresses the probability that carried clients entered service instantly on arrival. For large t we recognise that the *cdf* approaches one rather slowly, which indicates a heavy tailed distribution, which for overload is quite evident.

3.2.3 The matrix analytic approach to *Ph/Ph/1/s* queueing systems

In practice are the involved processes rarely perfect Poisson processes. Still, as already mentioned with infinite systems in section 3.1, these provide an analytically handy approximation and define the boundary between smooth and bursty processes. If an approximation by negative exponentially distributed events is insufficient we can approximate the true distributions of events by meshed negative exponentially distributed phases, the so called *phase type* distributions introduced in section 2.1 using the methods sketched in section 2.3, or the process itself by defining an according Markov Arrival Processes (MAP), introduced in section 2.1.3 and detailed in section 2.1.4.

Finite queueing systems composed of processes that can be expressed as Markov Arrival Processes (MAP), see section 2.1, can be numerically solved using the *Matrix Analytic Method* (MAM) similar to Neuts' *Matrix Geometric Method* presented in [73] and sketched in section 3.1.6. The Q-matrix of *Ph/Ph/1/s* queueing systems has block-diagonal form:

$$Q = \left[\begin{array}{c|c|c|c|c|c} B_1 & B_2 & 0 & \cdots & 0 & 0 \\ \hline B_0 & A_1 & A_2 & \ddots & \ddots & 0 \\ \hline 0 & A_0 & A_1 & A_2 & \ddots & \vdots \\ \hline \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \hline 0 & \ddots & \ddots & A_0 & A_1 & A_2 \\ \hline 0 & 0 & \cdots & 0 & A_0 & B_s \end{array}\right]$$

The major differences to MGM is that there exists a far end of the state transition diagram, an upper boundary-level B_s, here called the *blocking level*. This causes that we need not iteratively derive

the rates-matrix R because finite systems can always be represented by a finite Q-matrix, such that numerically $Q\,p = 0$ can be directly solved, applying the scheme presented in section 1.4.3.

To get the repeating level sub-matrices A_0, A_1, A_2 the same calculation as presented in section 3.1.6 is applicable,

$$\begin{aligned} A_0 &= I_A \otimes D_1(S) \\ A_1 &= D_0(A) \otimes I_S \,+\, I_A \otimes D_0(S) \\ A_2 &= D_1(A) \otimes I_S \end{aligned}$$

where $D_i(X)$ are the matrices defining the MAP processes, I_X are identity matrices of size according to the size of the indexed MAP process, and $\otimes$ indicates the Kronecker multiplication of matrices.

The lower boundary sub-matrices B_0, B_1, B_2 are composed of the transitions to, among, and from, idle states. When no customer is present the service process is interrupted, and therefore we only need to consider the parts that can be reached from levels above. In consequence are the B-matrices smaller than the A-matrices. In case of *Ph/Ph/...* we can calculate these from the involved process's $D_i(X)$ according to

$$\begin{aligned} B_0 &= \vec{\alpha}_A \otimes D_1(S) \\ B_1 &= D_0(A) \\ B_2 &= \vec{\alpha}_S^T \otimes D_1(A) \end{aligned}$$

where $\vec{\alpha}_X$ represents the entry-vector of the according MAP, and $\vec{\alpha}^T$ transposes the vector into a row. If blocked arrivals cause the same transitions as not blocked, just without increasing the number of customers present, than the upper boundary sub-matrix results as:

$$B_s = A_1 + A_2$$

In many cases it is easier to state the *Ph*-type definition $\vec{\mu}, \vec{\alpha}, q$ and calculate the MAP definitions. This can be done according to

$$\begin{aligned} D_0 &= T = (I_X \times \vec{\mu}) \times (q - I_X) \\ D_1 &= \vec{t}_0 \times \vec{\alpha} = (-T \times 1) \times \vec{\alpha} \end{aligned}$$

where T represents the transition-rate matrix, which results from the phase-rates vector $\vec{\mu}$ and the transition-probability matrix q, actually by multiplying each row i of q with the according phase-rate μ_i and inserting the negative phase-rate μ_i as diagonal elements. The exit-rate-vector $\vec{t}_0$ equals the positive row-sums of the transition-rate matrix T, which results from $\vec{t}_0 = -T \times 1$ whith 1 being a fittingly sized ones-matrix, and $\times$ indicating common matrix or vector multiplication as applicable.

This works out very nicely because all Q-matrix entries result directly from the *Ph*-type definitions (or the MAP definitions) of the involved processes. It is not necessary to explicitly set up the state transition diagram in order to find the boundary matrices. However, note that this is only possible if both processes defining the queueing system, being the arrival and the service process, are state independent. If one or both change in case the system is idle or completely filled, the boundary matrices B_x need to be composed according to the precise state transition diagram, correctly considering the process dependencies on the current state.

The following example illustrates the simplicity in case of state independent processes. We assume an arrival process that comprises an interrupted 2-phases Cox-generator, as depicted on the left in figure 3.43, and a 2-phases generalised-Erlang service process with for example a slow error recovery phase that partly causes a loop-back (re-transmission $\rightarrow$ infinite impulse response), depicted on the right in figure 3.43. The figure explicitly shows the *Ph*-type relevant, process internal,

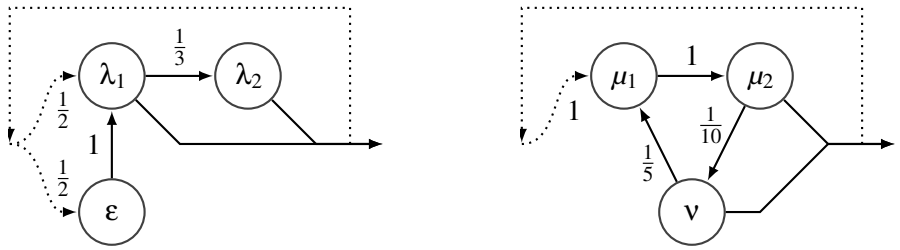


Figure 3.43: Example *Ph*-type arrival and service processes

transition probabilities q_{ij} and the entry probabilities α_i. These are rather few parameters, but they still define the assumed processes completely, i.e.,

$$\vec{\mu}_A = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \varepsilon \end{pmatrix} \quad \vec{\alpha}_A = \begin{pmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{2} \end{pmatrix} \qquad \vec{\mu}_S = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \nu \end{pmatrix} \quad \vec{\alpha}_S = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$q(A) = \begin{bmatrix} 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad q(S) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & \frac{1}{10} \\ \frac{1}{5} & 0 & 0 \end{bmatrix}$$

from which we get the rate based definitions

$$T(A) = \begin{bmatrix} -\lambda_1 & \frac{1}{3}\lambda_1 & 0 \\ 0 & -\lambda_2 & 0 \\ \varepsilon & 0 & -\varepsilon \end{bmatrix} \qquad T(S) = \begin{bmatrix} -\mu_1 & \mu_1 & 0 \\ 0 & -\mu_2 & \frac{1}{10}\mu_2 \\ \frac{1}{5}\nu & 0 & -\nu \end{bmatrix}$$

$$\vec{t}_0(A) = \begin{pmatrix} \frac{2}{3}\lambda_1 \\ \lambda_2 \\ 0 \end{pmatrix} \qquad \vec{t}_0(S) = \begin{pmatrix} 0 \\ \frac{9}{10}\mu_2 \\ \frac{4}{5}\nu \end{pmatrix}$$

and finally the MAP definitions

$$D_0(A) = \begin{bmatrix} -\lambda_1 & \frac{1}{3}\lambda_1 & 0 \\ 0 & -\lambda_2 & 0 \\ \varepsilon & 0 & -\varepsilon \end{bmatrix} \qquad D_0(S) = \begin{bmatrix} -\mu_1 & \mu_1 & 0 \\ 0 & -\mu_2 & \frac{1}{10}\mu_2 \\ \frac{1}{5}\nu & 0 & -\nu \end{bmatrix}$$

$$D_1(A) = \begin{bmatrix} \frac{1}{3}\lambda_1 & 0 & \frac{1}{3}\lambda_1 \\ \frac{1}{2}\lambda_2 & 0 & \frac{1}{2}\lambda_2 \\ 0 & 0 & 0 \end{bmatrix} \qquad D_1(S) = \begin{bmatrix} 0 & 0 & 0 \\ \frac{9}{10}\mu_2 & 0 & 0 \\ \frac{4}{5}\nu & 0 & 0 \end{bmatrix}$$

$$Q(A) = \begin{bmatrix} -\frac{2}{3}\lambda_1 & \frac{1}{3}\lambda_1 & \frac{1}{3}\lambda_1 \\ \frac{1}{2}\lambda_2 & -\lambda_2 & \frac{1}{2}\lambda_2 \\ \varepsilon & 0 & -\varepsilon \end{bmatrix} \qquad Q(S) = \begin{bmatrix} -\mu_1 & \mu_1 & 0 \\ \frac{9}{10}\mu_2 & -\mu_2 & \frac{1}{10}\mu_2 \\ \nu & 0 & -\nu \end{bmatrix}$$

required to construct the Q-matrix of the queueing system composed by these two processes.

Here both processes are defined by transition-matrices of size 3×3, and thus we get A_i matrices of size 9×9 and B_i matrices of size 9×3, 3×3, and 3×9, respectively. The blocking level's B_s matrix has the same size as A_i, which is evident as it results from $A_1 + A_2$.

Solving $Q\,p = 0$ with the condition $\sum p_{ij} = 1$, we get the state probabilities p_{ij} for all states. Summing over all state-probabilities multiplied with the number of present customers i we get the mean system filling, $\mathrm{E}[X] = \sum i\,p_{ij}$, and similarly the queue filling, $\mathrm{E}[Q] = \sum (i-1)\,p_{ij}$. Using Little's law $N = \lambda T$, considering that it is based on the arrivals actually entering the system $\lambda_{in} = (1 - Pb)\lambda$, we get the mean flow time $\mathrm{E}[T_f]$ and mean waiting time $\mathrm{E}[T_w]$, respectively. As the mean service time is the difference of the former two, $\mathrm{E}[T_h] = \mathrm{E}[T_f] - \mathrm{E}[T_w]$, we can use this to derive the *effective* service time of the *Ph*-type service process.

The *blocking probability* P_b results as the sum over all state probabilities in the blocking-level, $P_b = \sum_{i=s} p_{ij}$, and the *throughput* ϑ is the sum over all non-idle state probabilities times the mean service rate, $\vartheta = \bar{\mu} \sum_{i>0} p_{ij}$. Latter is not explicitly given in the *Ph*-type definition. However, we get $\bar{\mu} = \mathrm{E}[S]$ from the moments equation 2.6 stated in section 2.1.2.

Commonly, studies should be normalise to $\bar{\mu} = 1$. To achieve this the rates-vector of the service process needs to be scaled to $\vec{\mu}(1) = \mathrm{E}[S]\,\vec{\mu}$. In the same way we scale the arrival process's rates-vector to achieve different system loads ρ, i.e., $\vec{\lambda}(\rho) = \rho\,\mathrm{E}[A]\,\vec{\lambda}$. Note that here we switch from the general $mean_X \to \mu_X$ notation to the queueing system notation where we have $\mu_A \to \lambda$, $\mu_S \to \mu$.

To show calculated results and to compare these with simulation results, we need to select a system size, here $s = 9$, and state processes' phase-rates. For the results shown in figure 3.44 we assumed

$$\vec{\lambda} = \xi \begin{pmatrix} 3 \\ 1 \\ \frac{1}{2} \end{pmatrix} \qquad \text{and} \qquad \vec{\mu} = \zeta \begin{pmatrix} 1 \\ 2 \\ \frac{1}{2} \end{pmatrix}$$

as relations among phase rates. The actual service rates are normalised by a constant ζ to achieve $\bar{\mu} = 1$, and the arrival rates are scaled by ξ to yield the mean arrival rates shown on the x-axis in figure 3.44. We show several calculated metrics, and recognise that these look quite similar to those we already

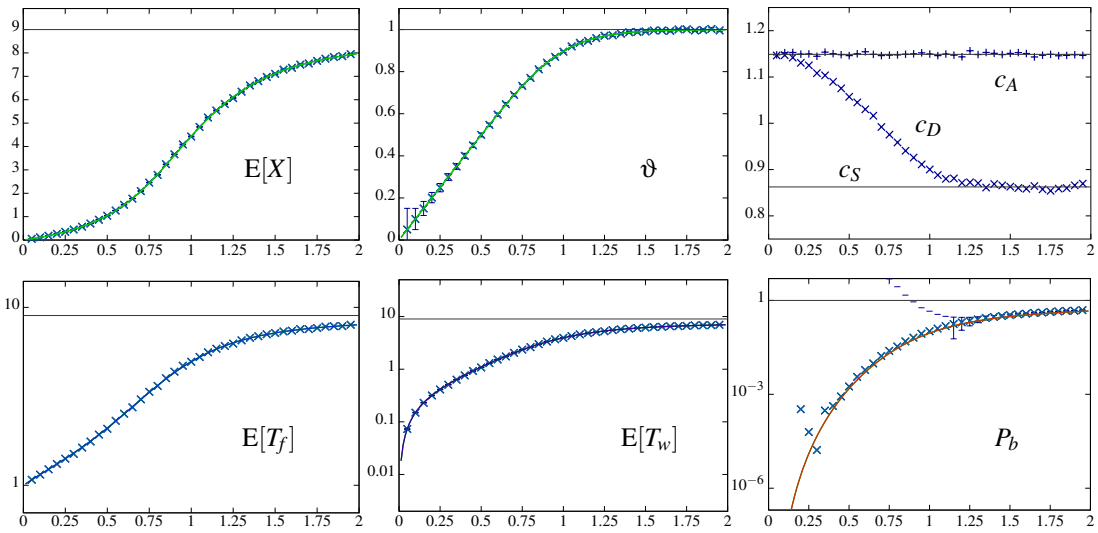


Figure 3.44: *Ph/Ph/1/9* example calculations (solid lines) and simulation results ($\times$)

know from *M/M/1/s*. Given the coefficient of variation of the arrival and service process used, this is no big surprise. However, looking more closely we notice that some curves are not as symmetric and smooth as usual, in particular the waiting time $\mathrm{E}[T_w]$ shows a quite steep bend toward zero load. Anyhow, this example shall express that this method yields precise results, not only an approximation, if the involved processes actually are *Ph*-type or MAP processes. It is up to the engineer to design and adjust system parameters in order to achieve a suitable system that balances loss-probability, response-time, server-utilization and waiting-time to the convenience of the client.

3.2.4 Finite population – the *M/M/n/s/c* model

Tore Olaus Engset presented in [79, 1918] an alternative queueing system where the number of potential clients c is not assumed to be infinite. In practice most client groups are finite, but often much bigger than the provided waiting space, such that they can be approximated as infinite.

However, if this is not the case, than the population bound c likely has some influence on the system performance. For example, if the system size s equals or exceeds the population size c, than no blocking can occur, $P_b(s \geq c) = 0$, even though the system is finite.

In this model the total arrival and service rates are both state dependent, at least for some states. If we assume $c \geq s \geq n$, the birth-death rates of the queueing system are

$$\lambda_i = \begin{cases} (c-i)\,\hat{\lambda} & 0 \leq i < s \\ 0 & i \geq s \end{cases} \qquad \mu_i = \begin{cases} i\mu & 0 \leq i \leq n \\ n\mu & i \geq n \end{cases} \tag{3.93}$$

where $\hat{\lambda}$ represents the arrival rate per *idle* client and μ the service rate per server. The according state transition diagram is depicted in figure 3.45.

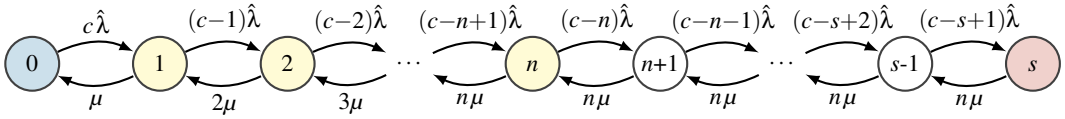


Figure 3.45: *M/M/n/s/c* state transition diagram

The steady state probabilities p_i for this system can be analytically calculated as presented by Kleinrock in [14, section 3.10.], briefly repeated here.

$$p_i = \begin{cases} p_0 \prod_{j=0}^{i-1} \frac{(c-j)\hat{\lambda}}{(j+1)\mu} = p_0 \binom{c}{i} \left(\frac{\hat{\lambda}}{\mu}\right)^i & 0 \leq i \leq n \\ p_0 \prod_{j=0}^{n-1} \frac{(c-j)\hat{\lambda}}{(j+1)\mu} \prod_{j=n}^{i-1} \frac{(c-j)\hat{\lambda}}{n\mu} = p_0 \binom{c}{i} \left(\frac{\hat{\lambda}}{\mu}\right)^i \frac{i!}{n!} n^{n-i} & n \leq i \leq s \end{cases} \tag{3.94}$$

The analytic expression for p_0 is rather bulky. However, for a given example we can numerically calculate all p_i even without an explicit equation for p_0.

Two steps p_i calculation

• initially assume	$p_0^* = 1$	apply 3.94 $\forall_i$	to get all p_i^*
• for $\sum p_i = 1$ set	$p_0 = \frac{1}{\sum p_i^*}$	and finally get	$p_i = p_0 p_i^*$

(3.95)

Based on this calculation scheme we can numerically calculate the mean system filling $\mathrm{E}[X] = \sum i p_i$ and queue filling $\mathrm{E}[Q] = \sum (i-n)^+ p_i$. Using Little's law $N = \lambda T$ we get from these the mean flow time $\mathrm{E}[T_f]$ and waiting time $\mathrm{E}[T_w]$. However, to do this we first need to find the *ingress rate* λ_{in}, which equals the *throughput* ϑ because all customers that enter the system become served some time. This and the global arrival rate λ_s can as well be calculated numerically from the steady state probabilities p_i based on the state transition diagram depicted in figure 3.45.

$$\lambda_s = \hat{\lambda} \sum_{i=0}^{s} (c-i)\, p_i = c\hat{\lambda} - \hat{\lambda} \sum_{i=0}^{s} i p_i = \hat{\lambda}\,(c - \mathrm{E}[X])$$

$$\lambda_{in} = \hat{\lambda} \sum_{i=0}^{s-1} (c-i)\, p_i = \lambda_s - \hat{\lambda}\,(c-s) p_s = \hat{\lambda}\big(c - \mathrm{E}[X] - (c-s) p_s\big)$$

$$\vartheta = \mu \sum_{i=1}^{n} i p_i + n\mu \sum_{i=n+1}^{s} p_i = \mu\Big(\sum_{i=0}^{n} i p_i + n\big(1 - \sum_{i=0}^{n} p_i\big)\Big) = n\mu\Big(1 - \sum_{i=0}^{n} \frac{n-i}{n} p_i\Big)$$

Using the results for λ_s and λ_{in} with the relation $\lambda_{in} = \vartheta = \lambda_s (1 - P_b)$ we get an equation for P_b.

$$P_b = \frac{(c-s)p_s}{c - \mathrm{E}[X]} \tag{3.96}$$

Equation 3.96 is applicable for *M/M/n/s/c* only, as figure 3.48 shows. Still, it requires the numerical calculation of all steady state probabilities p_i using for examples the scheme 3.95 to get the mean system filling $\mathrm{E}[X] = \sum i p_i$, and explicitly the probability p_s for the system being full. For the general case $n < s < c$ there exists to our best knowledge no analytic equation to calculate P_b, $\mathrm{E}[X]$, $\mathrm{E}[Q]$, $\mathrm{E}[T_f]$ or $\mathrm{E}[T_w]$ that does not require calculating the steady state probabilities p_i first. However, analytic equations applicable for special cases exist, and are presented shortly.

Different arrival generation process

The *different arrival rate* $\hat{\lambda}$ *definition* needs to be considered when comparing the Engset setting with the conventional setting assuming infinite customer populations (Erlang setting). With finite customer populations the arrival generation rate $\hat{\lambda}$ is commonly stated *per customer*, and is *conditioned to not being in the system*, meaning that it applies for *idle* customers only (idle time distribution). This is straightforward because the finite customer population model assumes that customers being in the system, currently *busy*, do not generate new arrivals until they left the system, become idle again. In figure 3.46 we sketch this arrival process over time per customer, the customers state model in total, and the state flow diagram that results per customer. Note that in latter the state index j refers to the

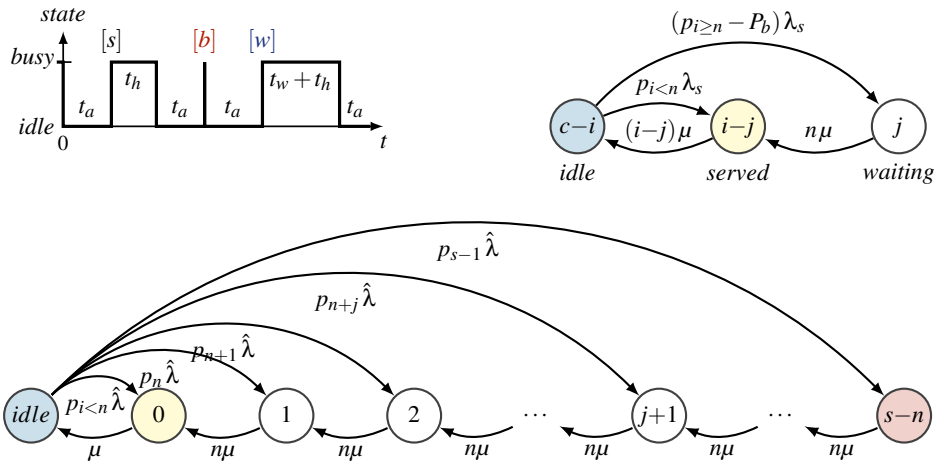


Figure 3.46: Arrival generation process commonly assumed with finite population queueing models

customer's position in the queue, not considering any customers queued in behind. The core problem with this model is that the arrival process depends on the steady state probabilities of the queueing system and that these themselves depend on the arrival process. Every accepted arrival reduces the momentary arrival rate because the entering customer does not generate new arrivals while in the system. In consequence is the global *arrival process not memoryless*.

The mean arrival rate is upper bounded by $c\hat{\lambda}$, the rate at which arrivals occur when no arrival is accepted ($s = 0$). A lower bound for the mean rate results vice versa for the case where all arrivals are accepted, $P_b = 0$. In that case is $\lambda_s = \vartheta$, and we can state for the actual mean arrival rate and load

$$\vartheta \leq \lambda_s \leq c\hat{\lambda} \qquad\qquad \rho_{carried} \leq \rho_{offered} \leq c\,\hat{\rho}_{idle}$$

where $\hat{\rho} = \frac{\hat{\lambda}}{\mu}$ states the load that every customer would generate if no arrival is accepted. Interesting to note, the more arrivals are rejected, the more load is generated by the finite customer population. This models many practical cases better than the Erlang setting does.

A problem occurs if we only know the actual arrival rate to the system λ_s, and the population size c that caused this, for example from measurement. In this case we need to find the *hidden* $\hat{\lambda}$ by iteration:

1. initially assume $\mathrm{E}[X]^*=0$
2. calculate approximate arrival generation rate $\hat{\lambda}^* = \dfrac{\lambda_s}{c-\mathrm{E}[X]^*}$
3. solve 3.94 using algorithm 3.95 to get all p_i^*
4. calculate new $\mathrm{E}[X]^*=\sum i p_i^*$
5. if $(\Delta_{\mathrm{E}[X]}>\Delta_{max})$ return to 2^{nd} step until $\mathrm{E}[X]$ is sufficiently stable
6. use the just calculated p_s^* and $\mathrm{E}[X]^*$ with equation 3.96 to get the blocking probability P_b

Using this iterative approach we find the load dependent relation between the arrival rate to the system λ_s and the arrival generation rate $\hat{\lambda}$ depicted in figure 3.47. The iteration converges quite

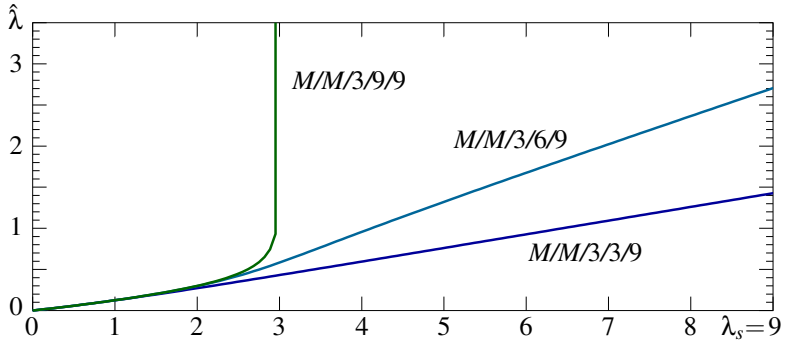


Figure 3.47: Arrival generation rate $\hat{\lambda}$ over offered load λ_s ($\mu=1$)

well, typically in less than 25 steps for $\Delta_{max}=10^{-9}$. The most steps occur around $\lambda_s=n\mu$. Overload, $\lambda_s\geq n\mu$, cannot occur with the loss-less *M/M/n/c/c* system. Here grows $\hat{\lambda}\to\infty$ for $\lambda_s\to n\mu$. The system properties $\mathrm{E}[X]$ and P_b over $\hat{\rho}$ are shown in figure 3.48, together with simulation results.

Special cases

• *M/M/n/n/c*: In case the system has no queue, $s=n$, it becomes a pure loss system. This is the finite customer population sibling to the Erlang loss system.

$$P_b = \frac{\binom{c-1}{n}\hat{\rho}^n}{\sum_{j=0}^{n}\binom{c-1}{j}\hat{\rho}^j} \qquad p_i = \frac{\binom{c}{i}\hat{\rho}^i}{\sum_{j=0}^{n}\binom{c}{j}\hat{\rho}^j} \qquad \text{in case } s=n\leq c$$

Queue-less systems alike are discussed later on in section 3.3. The *Engset loss formula* for P_b and *Engset distribution* for p_i are stated here for completeness only.

• *M/M/n/c/c*: In case the system size equals (or exceeds) the customer population size, $s\geq c$, the system becomes a loss-less queueing system. Every customer can be accommodated at any time.

$$P_b = 0 \qquad p_i = \text{use 3.95 to solve 3.94} \qquad \text{in case } c\leq s$$

The steady state probabilities p_i and the system performance can be calculated as in the general case. System states with an index $i>c$ have zero probability and thus no relevance.

• *M/M/c/c/c*: In case the number of servers equals (or exceeds) the population size, $n\geq c$, no queueing occurs and only the first part of equation 3.94 applies.

$$P_b = 0 \qquad p_i = \frac{\binom{c}{i}\hat{\rho}^i}{(1+\hat{\rho})^c} \qquad \text{in case } c\leq n\leq s$$

This system represents the finite sibling to the infinite server model (*M/M/*∞), a system with no queue, $s=n$, and $P_b=0$. It is discussed in more detail in section 3.3.2.

In figure 3.48 we show simulation results together with calculated results (continuous lines). The system capacity $n\mu$ and the population size c is the same for all. The only changing parameter is the queue size $s-n$. The properties and insensitivities of such systems are revealed by assuming different idle time and holding time distributions. First we notice that the mean system filling $\mathrm{E}[X]$

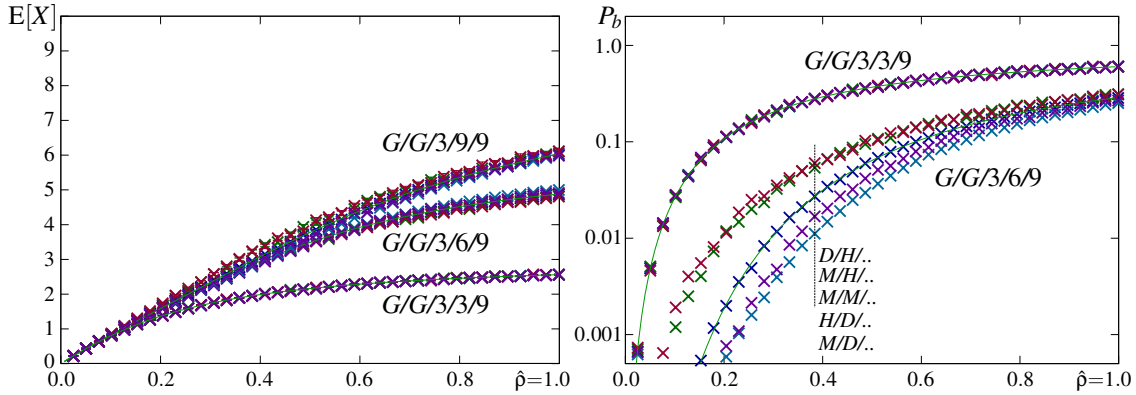


Figure 3.48: *G/G/n/s/c* $\mathrm{E}[X]$ and P_b for different idle and holding time distributions (M=Markovian(c_v=1), D=deterministic(c_v=0), H=hyper-exp.(c_v=5), μ=1, samples/point=200.000)

hardly differs for different arrival and service time distributions. For the loss system *G/G/3/3/9* it is least and perfectly identical for all studied examples. The blocking probability P_b(*G/G/3/3/9*) is also identical for all process pairings, and worst compared to other queue sizes. For a system size equal the population size, *G/G/3/9/9*, no blocking occurs, $P_b=0$, and the system filling $\mathrm{E}[X]$ is maximal. In the general case *G/G/3/6/9* the system filling $\mathrm{E}[X]$ and the blocking probability P_b are both in between the extremes defined by the other systems. The blocking probability P_b calculated via equation 3.96 matches the simulation results we get for the *M/M/3/6/9* system. In the other simulation results we recognise that the blocking probability P_b is less for deterministic service times (*G/D/3/6/9*) and considerably worse for hyper-exponential service times (*G/H/3/6/9*). With deterministic service times we recognise also a slight dependence on the idle time distribution. The blocking probability P_b is slightly higher with hyper-exponential idle times (*H/D/3/3/9*) compared to negative exponentially distributed idle times (*M/D/3/3/9*).

The special case $c \leq n$ causing the *M/M/c/c/c* case is not shown in figure 3.48. It resembles the bounded version of the *M/M/*∞-system presented in section 3.3.2. See figure 3.56 and figure 3.59 for the comparison of *M/M/*∞ with *M/M/c/c/c*.

That with the Engset setting *a client's load share does not contribute to the arrival rate while the client is in the system* allows us to use *M/M/n/s/c* models to more accurately model the load that enters a network node when analysing the transmission related queueing process (line buffers). For serial packet reception the maximum number of simultaneous arrivals is upper bound by the number of ingress ports. While a packet is received the according port is effectively blocked from causing more arrivals to the node. The more load enters a node via a particular port, the more influences this effect the distribution of the load contributed by that port. This commonly improves the node's performance; thus it is called the *streamlining effect*, discussed for example in [56, 80]. This property explains the negligible influence of ingress line buffers when used for local re-clocking. Due to $c=1$ equals the buffer's egress distribution its ingress distribution, only some in average negligible time-lag is caused.

3.3 Queue-less systems

This section covers some system models that on a first glance apply to other fields rather than packet based data communication. For packet switching the queue is essentially needed to realise *statistical multiplexing*, being the core mechanism that makes packet switching simple and sufficiently efficient. Still, many links connecting packet switched nodes are provided by circuit switched technologies, mainly SDH and OTN, particularly where Ethernet fails to span the geographic distance between nodes. Circuit switching is commonly modelled by loss systems. On the other side, above the layer 4 connection, we commonly find *logical* connections that belong to different sessions but share the available end-to-end capacity. Their interplay can be modelled by *processor sharing* models, where the server represents the abstracted layer 4 connection between two end-systems (client-to-server, server-to-server, or even peer-to-peer).

In between these two we discuss rather academic systems that either assume an infinite number of servers or are quite over-dimensioned. These are useful to approximate loss-less systems with many servers, to integrate stochastic delays in queueing network analysis, and they provide upper bounds and benchmarks for realistic systems.

3.3.1 Loss systems – *M/G/n/n*

The functional contribution of the *generalised multi protocol switching* (GMPLS) control hierarchy is the option to change the circuit switched lower layer connectivity on demand. Traditionally, transmission resources are not provided dynamically to the network layer. Thus, a capable lower layer network technology is required to smartly, meaning efficiently and demand specific, utilise the available network resources. However, temporarily a requested connectivity may be unachievable. *Loss models* can be used to assess the likelihood that packet nodes can be connected on demand, based on the request-arrival distribution and the life-time distribution of these connections, which finally contribute the fundamental data transmission among neighbouring nodes.

In case of regular topologies it is sometimes possible to model an entire data transmission network by a single loss system. For example, a symmetric bidirectional 1:1-protected double ring topology can be represented by a single multi-server system, as shown in figure 3.49. This is possible here

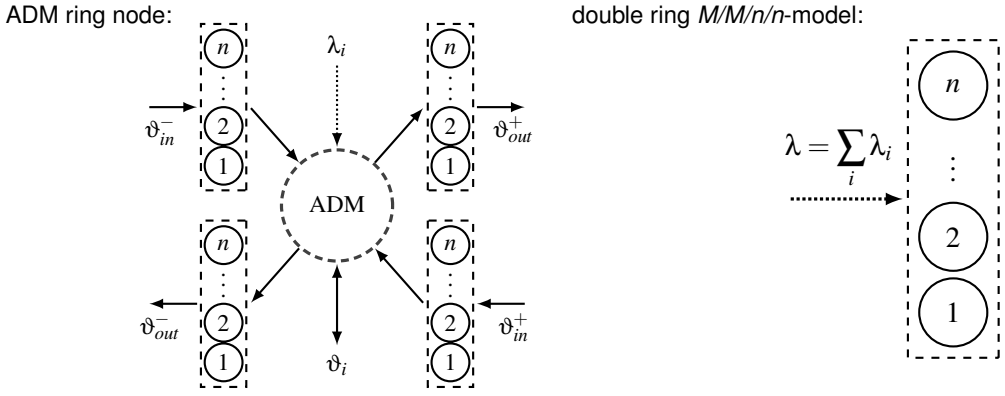


Figure 3.49: Add-drop-multiplexing ring node (ADM) with adjacent links of a circuit switched double ring network (left) and the *M/M/n/n*-model that results for 1:1-protected connections and symmetric, bidirectional capacity assignment (right)

because every connection occupies the demanded capacity on every single link along the entire ring: in one orientation to establish the working path, in the other to establish/provide the protection path. Thus, the number of servers equals the capacity units available on the links connecting the ring-nodes, precisely their bottleneck if capacities vary. The example shows that quite complex structures may be reducible into rather simple models if system symmetries enable it or may be assumed.

The state transition diagram for the *M/M/n/n* loss system is shown in figure 3.50, where we assume negative exponentially distributed times in between connection requests ($t_a = 1/\lambda$) as well as for the connection holding times ($t_h = 1/\mu$).

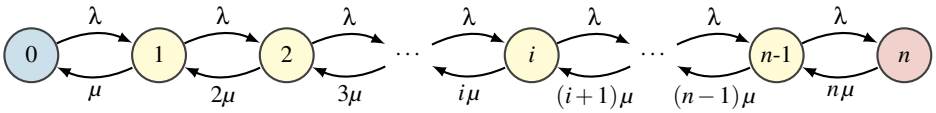


Figure 3.50: *M/M/n/n* state transition diagram

The analytic solution of this queueing system goes back to *A.K. Erlang* [66, 1917], and yields the *Erlang_B* equation, or simply *Erlang loss formula* $E_B(\rho, n)$.

$$p_i = \frac{\frac{\rho^i}{i!}}{\sum_{j=0}^{n} \frac{\rho^j}{j!}} \qquad P_b = E_B(\rho, n) = \frac{\frac{\rho^n}{n!}}{\sum_{j=0}^{n} \frac{\rho^j}{j!}} \qquad \frac{1}{E_B(\rho,k)} = 1 + \frac{k}{\rho\, E_B(\rho, k-1)} \tag{3.97}$$

Erlang_B yields the *blocking probability* P_b from the offered load ρ defined in Erlang ($\rho = \frac{\lambda}{\mu}$) and the provided number of equally performing resources n. Numerically, this is best calculated iteratively, starting with $E_B(\rho, 0) = 1$ and using the inverse $\frac{1}{E_B}$ for stability, as shown on the right in 3.97.

The solution relates the performance and resource demand of systems where a finite pool of identical resources defines the system capacity $n\mu$, and where requests cannot wait, meaning that on arrival they need to be served or rejected instantly, such that no queue may build up ($s = n$).

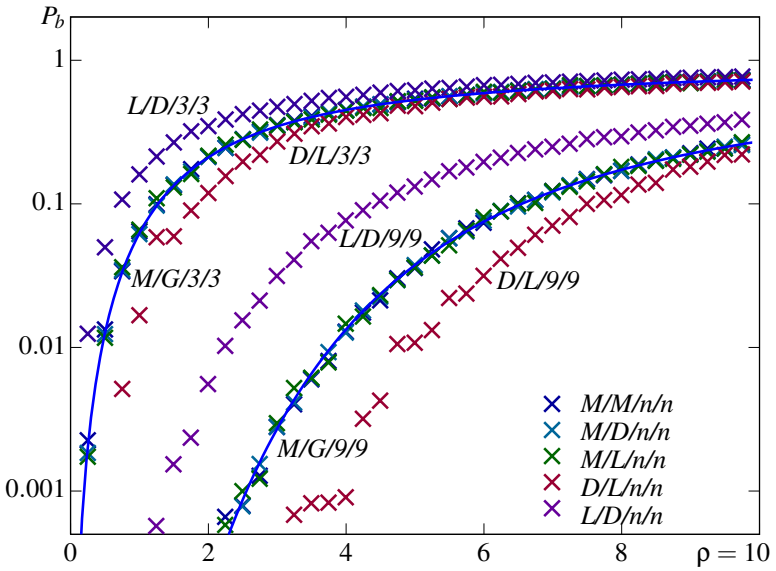


Figure 3.51: Blocking probability P_b over load for different *G/G/n/n* systems
(M=Markovian, D=deterministic, L=Lomax(ρ, 2), n=3, 9, μ=1, samples/point=50.000)

The blocking probability P_b calculated using equation 3.97 is shown in figure 3.51 (continuous line) together with simulation results for different *G/G/n/n* models. Due to PASTA, Poisson arrivals see time averages, is for Markovian arrivals the P_b of loss systems independent of the actual service time distribution T_h. Accordingly, simulation results for different service processes superpose each other for $A = M$, and thus, the *Erlang_B* formula (3.97) *is applicable for any M/G/n/n loss-system.*

For smooth arrivals P_b is less, and for heavy tailed arrivals it is worse. In figure 3.51 we actually show the cases where one is deterministic and the other Lomax(ρ, 2) distributed, with coefficient of variation 0 and ∞, respectively. As PASTA is not applicable in these cases, the results are not independent of the service process and thus represent the best and worst results for the worse and

better case. Evidently, for *D/D/n/n* we would get zero blocking for $\lambda < n\mu$, and for *L/L/n/n* we would get worse mean blocking than for the *L/D/n/n* case shown.

Finally we note that here P_b is the prime performance metric because for any *G/G/n/n* system the flow time equals the service time, $T_f = T_h$, and is thus independent of the only adjustable system parameter n. However, if we would keep the system capacity $\mu_s = n\mu$ constant we would introduce system size dependent mean holding times $\mathrm{E}[T_h] = \frac{n}{\mu_s}$, and thus mean flow times $\mathrm{E}[T_f]$ that increase linearly with n. The mean system filling $\mathrm{E}[X]$ depends primarily on the mean load $\rho = \frac{\lambda}{\mu}$, and is rather independent of the involved service process.

Evident but still notable, the blocking probability $P_b^{G/G/n/n}$ improves (decreases) with increased system size n, even if we keep the system capacity $n\mu$ constant. This is based on the simple, time and load invariant phenomenon that the more servers are available the less probable are all these occupied at the same time. Thus, to minimise the loss rate λP_b it is always best to split the required system capacity $n\mu$ to as many servers as possible.

Loss system design

Without a queue, the mean service rate μ per server defines the mean flow time, and thus, to design a system that provides a certain mean flow time $\mathrm{E}[T_f]$ with a given blocking probability P_b, we need to first calculate μ and than the number of servers n that is required to handle a targeted arrival rate λ.

$$\mu = \frac{1}{\mathrm{E}[T_f]} \qquad \rightarrow \qquad \rho = \frac{\lambda}{\mu} \qquad \rightarrow \qquad n = \min_n(n | E_B(\rho, n) \leq P_b)$$

Latter is easily achieved using the recursive calculation of $E_B(\rho, n)$ stated in equation 3.97. Applying better serves with higher rate can reduce the number of required servers, but demands a higher system capacity $n\mu$, which commonly causes higher implementation and operation expenses.

Finite customer population – *M/G/n/n/c*

In section 3.2.4 we first introduced finite customer populations c. In practice all populations are finite, in particular the number of ingress ports at network nodes. However, models assuming an infinite population fit well if the mean service rate is far below the maximum ingress rate. This is commonly the case for packet transmit queues at egress ports if the ingress port count is sufficiently high. Still, at switches that perform *cut-through switching* no new switching requests can origin from a currently occupied ingress port. Cut-through switches, where queues are inexpedient, are therefore a technology where the *M/G/n/n/c* model has to be applied in order to achieve accurate modelling.

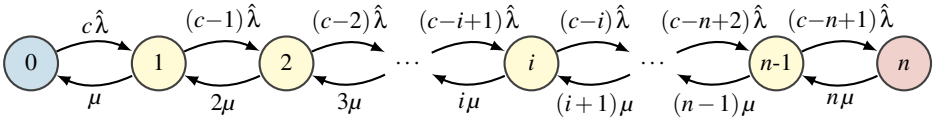


Figure 3.52: *M/M/n/n/c* state transition diagram

Figure 3.52 depicts the state transition diagram of the loss system with finite customer population. The difference to the infinite population case, depicted in figure 3.50, is the system state dependently decreasing arrival rate. Sometimes this system is called *truncated binomial model*, in reference to the binomial coefficients in its steady state distribution. We use the Kendall notation to be consistent.

As already stated in section 3.2.4 is with finite customer population models the *offered load* defined *per customer*, using $\hat{\lambda}$ and $\hat{\rho} = \frac{\hat{\lambda}}{\mu}$. The parameters state the *arrival generation rate* (mean idle period distribution) and a kind of *maximum Erlang load per customer*. Both refer to the load that a customer would only generate if all arrivals are rejected. Every accepted arrival interrupts

this arrival generation because the customer is assumed to not generate new arrivals while he is in the system. This reduces the effective arrival rate and also causes that the global arrival process is not memoryless, even if the processes used to model the arrival generation and holding time are Markovian. Please refer to section 3.2.4 for a detailed discussion of the peculiarities of the *different arrival generation process* that results from the Engset setting.

For loss system with finite customer populations T.O. Engset published in [79, 1918] the steady state probability p_i distribution, called the *Engset distribution*, and the equation to calculate the blocking probability P_b, called *Engset loss formula* Eng_B in analogy to Erlang's loss formula [81].

$$p_i = \frac{\binom{c}{i}\hat{\rho}^i}{\sum_{j=0}^{n}\binom{c}{j}\hat{\rho}^j} \qquad P_b = Eng_B(\hat{\rho},n,c) = \frac{\binom{c-1}{n}\hat{\rho}^n}{\sum_{j=0}^{n}\binom{c-1}{j}\hat{\rho}^j} \tag{3.98}$$

$$\frac{1}{Eng_B(\hat{\rho},k,c)} = 1 + \frac{k}{\hat{\rho}(c-k)\,Eng_B(\hat{\rho},k-1,c)} \tag{3.99}$$

We notice that $P_b \neq p_n$. This is evident because if a customer is not in the system, and here this is necessary to cause an arrival, the probability that the system is full when the arrival occurs is that of the system being full without that customer in its population. PASTA is not applicable because due to the finite population the arrival process to the system is not memoryless, even if per customer both the idle and the service time distribution are negative exponentially distributed.

There again exists a recursive calculation option that looks quite similar to that in the Erlang setting. It simply replaces the factor ρ by $\hat{\rho}(c-k)$ in the multiplication with the previous result for $k-1$ servers. As common, the recursion starts with $Eng_B(\hat{\rho},0,c)=1$, and is perfectly suited to find the number of servers n required to reach a certain blocking probability P_b target.

Deriving an analytic equation for the system filling from $\mathrm{E}[X] = \sum i p_i$ using equation 3.98 would be straightforward, but requires some mathematical tricks [14]. Here we can actually find it much easier exploiting the specifics of the loss system. No queue exists, and accordingly equals the system filling, expressed by the system state index i, the number of currently busy servers. Using the state transition diagram shown in figure 3.52 we get

$$\vartheta = \sum_{i=0}^{n} i\mu p_i = \mu\,\mathrm{E}[X] \qquad \lambda_s = \hat{\lambda}\sum_{i=0}^{n}(c-i)\,p_i = c\,\hat{\lambda} - \hat{\lambda}\sum_{i=0}^{n} i p_i = \hat{\lambda}(c-\mathrm{E}[X])$$

where ϑ is the system throughput, and λ_s the *effective arrival rate to the system*. By $\vartheta = \lambda_s(1-P_b)$ the two calculations become joined and we get:

$$\mathrm{E}[X] = \frac{c\,\hat{\lambda}(1-P_b)}{\mu+\hat{\lambda}(1-P_b)} = \frac{c\,\hat{\rho}(1-P_b)}{1+\hat{\rho}(1-P_b)} \qquad \lambda_s = \frac{c\,\hat{\lambda}}{1+\hat{\rho}(1-P_b)} \qquad \vartheta = \frac{c\,\hat{\lambda}(1-P_b)}{1+\hat{\rho}(1-P_b)} \tag{3.100}$$

The idea in behind the Engset setting can be explained by the so called *intended load* $\tilde{\rho}$: per customer exactly one arrival occurs per arrival-service cycle if $P_b=0$. The mean length of this cycle is t_a+t_h, where $t_a=\frac{1}{\tilde{\lambda}}$ and $t_h=\frac{1}{\mu}$. Its inverse is the *intended arrival rate* $\tilde{\lambda}$ per customer, and divided by the mean service rate μ we get the indented load $\tilde{\rho}$ per customer:

$$\tilde{\lambda} = \frac{1}{t_a+t_h} = \frac{\hat{\lambda}}{1+\hat{\rho}} \qquad \tilde{\rho} = \frac{\hat{\rho}}{1+\hat{\rho}} \qquad \vartheta \leq c\,\tilde{\lambda} \leq \lambda_s \leq c\,\hat{\lambda}$$

Assuming constant intended system load $c\,\tilde{\rho}$ we recognise that the higher the blocking probability P_b is, the less load can pass the system, but also the more load is generated by the finite number of sources. This increase is the response of the traffic sources to a system failing to serve all arriving loads.

In case we only know the effective arriving rate λ_s, for example from measurements, but not the intended load per source $\tilde{\rho} = \frac{\tilde{\lambda}}{\mu}$ or the mean idle time $\frac{1}{\tilde{\lambda}}$, than we need to find the *hidden* $\hat{\lambda}$ by successive iteration in order to calculate the system properties [81].

1. initially assume $P_b^* = 1$
2. calculate approximate arrival generation rate $\hat{\lambda}^* = \dfrac{\lambda_s}{c - \frac{\lambda_s}{\mu}(1 - P_b^*)}$
3. solve $Eng_B(\frac{\hat{\lambda}^*}{\mu}, n, c)$ using 3.99 to get new P_b^*
4. if $(\Delta_{P_b} > \Delta_{max})$ return to 2^{nd} step until P_b is sufficiently stable

Using this iterative approach we find the load dependent relation between the arrival rate to the system λ_s and the arrival generation rate $\hat{\lambda}$ depicted in figure 3.53. The iteration converges quite well,

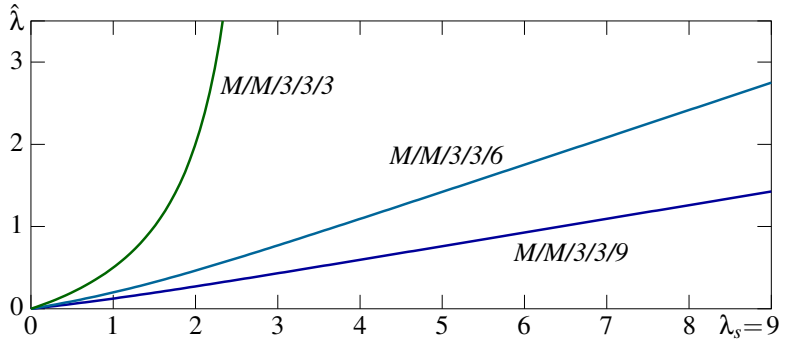


Figure 3.53: Arrival generation rate $\hat{\lambda}$ over offered load λ_s (μ=1)

typically in less than 20 steps for Δ_{max}=10^{-9}. Overload, $\lambda_s \geq n\mu$, cannot occur with the loss-less *M/M/c/c/c* system, here grows $\hat{\lambda} \to \infty$ for $\lambda_s \to n\mu$. The more customers exist, in relation to the number of servers, the less exceeds $c\,\hat{\lambda}$ the actual arrival rate λ_s.

The upper two graphs in figure 3.54 show the system properties $\mathrm{E}[X]$ and P_b over $\hat{\rho}$, calculated via equation 3.100 and 3.98 respectively (solid lines), together with simulation results for *M/M/..* , *M/D/..* , *M/H/..* , *H/D/..* , *D/H/..* finite source loss systems. The system filling $\mathrm{E}[X]$ confirms that the less customers exist, the slower rises the system filling. This is not that visible if we look at the lower graph showing $\mathrm{E}[X]$ over the actually offered load λ_s (μ=1$\to\rho\equiv\lambda$). Here we see that for the loss-less *G/G/3/3/3* system the system filling equals the load, $\mathrm{E}[X] = \frac{\lambda_s}{\mu}$, up to the system being filled up at $\mathrm{E}[X] = 3$. Seemingly, and quite practically, provides the system filling we get for the infinite customer population $\mathrm{E}\left[X^{M/G/3/3}\right]$ a lower bound (dashed line).

For the loss-less *G/G/3/3/3* system we get P_b=0, and thus we have in the graphs on the right only results for population size $c > n$. In the upper graph we find the simulation results sitting perfectly on the calculated curves (solid lines), actually for any process pairing. The thin densely dashed lines refer to equation 3.101 and belong to a different setting, sketched shortly. The lower right graph shows P_b over λ_s, and again are the differences for c=6 and c=9 marginal in this representation. Here, the dashed line showing $P_b^{M/G/3/3}$ provides now an upper bound. Thus, if the traffic actually offered to the system λ_s is known, we can savely use the Erlang setting, but schould consider that in case the number of sources is very small, the Erlang model yields somewhat conservative results.

That the blocking probability P_b does not depend on the holding time process T_h might have been expected from the infinite Erlang case. That for the Engset setting both distributions have no influence on the mean system filling $\mathrm{E}[X]$ nor on the blocking probability P_b is quite an astonishing result. From this finding we conclude without proove that the *Engset loss model is insensitive to both, the arrival generation and holding time distribution*. The equations and procedures found assuming *M/M/n/n/c* are thus *good for the general G/G/n/n/c model* [30, 8.3].

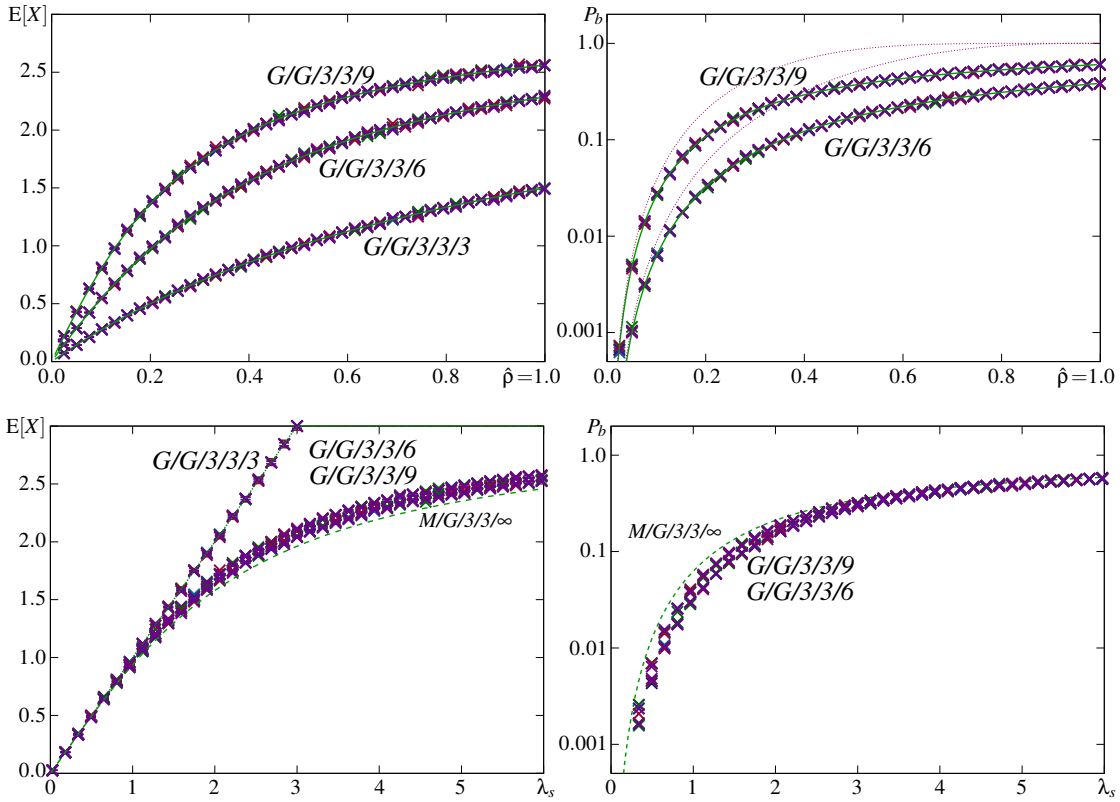


Figure 3.54: E[X] and P_b over $\hat{\rho}$ and λ_s for different *G/G/n/n/c* systems and population sizes c (M=Markovian(c_v=1), D=deterministic(c_v=0), H=Hyper-exp.(c_v=5), n=3, μ=1, samples/point=200.000)

In the literature we find another formula that somehow relates to this system, the so called *Binomial finite source loss formula* [82]:

$$P_b = \sum_{i=n}^{c-1} \binom{c-1}{i} \tilde{\rho}^i (1-\tilde{\rho})^{c-1-i} \tag{3.101}$$

This other P_b calculation assumes that the arrival generation process does not depend on the acceptance of arrivals. It demands that all customers behave as if being served, which natively occurs when load is blocked without telling the source. Seemingly unrealistic, this is perfectly common with optical burst switching (OBS) and datagram services alike UDP. With this setting the actually offered load equals the intended load, $\rho_s = c\,\tilde{\rho}$.

The higher P_b in figure 3.54 results from carelessly setting $\hat{\rho} = \tilde{\rho}$. If we consider $\hat{\rho} = \frac{\tilde{\rho}}{1-\tilde{\rho}}$ the blocking probability P_b occurring with this setting cannot be higher than that found for the Engset setting, simply because in general less load is offered, $c\,\tilde{\rho} \leq \rho_s \leq c\,\hat{\rho}$.

3.3.2 Infinite capacity – *M/G/∞*

The *M/G/∞*-system is the infinite version of the *M/G/n/n*-system discussed above. Practically this cannot be realised because an infinite number of servers, each providing a finite capacity, results in a system offering infinite capacity. In the real world this is impossible, but mathematically we can assume and solve such a system. The results yield some interesting insight and can at least be used to analyse systems with seemingly infinite service capacity.

The state transition diagram for the *M/M/∞* models is shown in figure 3.55. Solving this, we find that the steady state probabilities p_i are Poisson distributed with mean $\mathrm{E}[X] = \rho = \frac{\lambda}{\mu}$.

$$p_i = \frac{\rho^i}{i!}\, e^{-\rho} \tag{3.102}$$

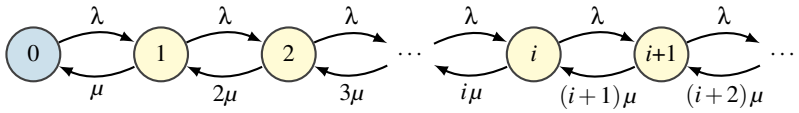


Figure 3.55: *M/M/∞* state transition diagram

Again, the steady state probabilities do not depend on the actual service time distribution F_{T_h}, only on its mean rate μ. Thus, all performance metrics that can be directly calculated from the steady state probabilities p_i are generally applicable for any *M/G/∞* system.

Applying Little's law $N=\lambda T$ we get $\mathrm{E}[T_f] = \frac{1}{\mu} = \mathrm{E}[T_h]$, which is evident because with infinitely many servers no client ever needs to wait for service. Being an infinite system, blocking cannot occur ($P_b=0\ \forall_\rho$). Properties of particular interest are the response to non Markovian arrival processes and the relation among arrival, service and departure process. The mean system filling $\mathrm{E}[X]$ for different inter-arrival and service time distributions is depicted in figure 3.56 together with the variation coefficient of departures c_D over increasing load. We recognise that the mean system filling $\mathrm{E}[X]$,

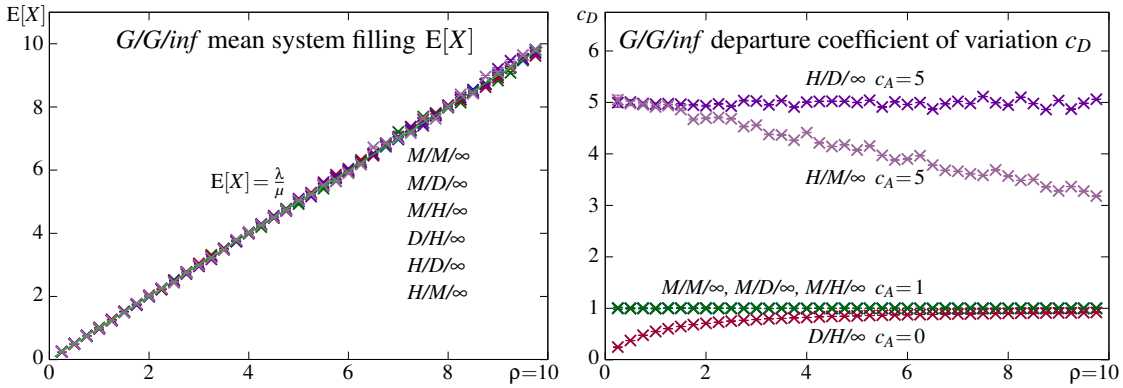


Figure 3.56: Mean system filling $\mathrm{E}[X]$ and departure coefficient of variation of *G/G/∞* systems (M=Markovian(c_v=1), D=deterministic(c_v=0), H=hyper-exp.(c_v=5), μ=1, samples/point=200.000)

shown in the left sub-figure, is independent of the process characteristics, actually of both, the arrival and the service process. Only higher moments are affected by the involved distributions.

The departure variation coefficient c_D, shown on the right, indicates that for Markovian inter-arrival times the departure distribution remains Markovian. Independent of the service distribution $F_{T_h}(t)$ we find c_D=1 for *M/M/∞*, *M/D/∞*, and *M/H/∞*. In case of deterministic arrivals, here *D/H/∞*, the departure distribution asymptotically approaches a coefficient of variation c_D=1 from below, even for hyper-exponential service time distribution, whereas for hyper-exponential distributed arrivals and Markovian service, *H/M/∞*, it approaches c_D=1 from above.

This effect is more clearly visible in figure 3.57, where we show the inter-departure time histograms of the same *G/G/∞* examples. At load $\rho=1$ only one server is in average busy and the arrival distribution dominates the departure distribution. Only for deterministic arrivals causes the holding time distribution F_{T_h} a clearly visible change, but the peak at T_D=1 is still dominant (*D/H/∞*). At an increased load of $\rho=9$ we recognise that most departure distributions approach the negative exponential distribution. The peak from deterministic arrivals entirely vanishes, the correlation among arrivals is nearly entirely removed by the parallel service processes with uncorrelated and rather bursty holding times (c_B=5). Contrary thereto, for deterministic holding times the departure distribution equals the arrival distribution (*H/D/∞*), independent of the number of involved servers, because all departures occur just time shifted by a constant factor.

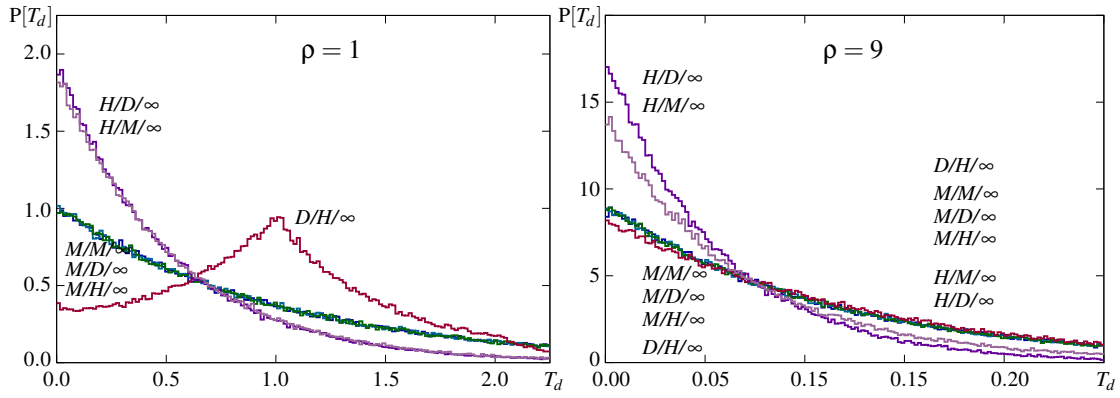


Figure 3.57: Inter-departure time histograms $P[T_d]$ for the examples shown in figure 3.56 (at load $\rho = 1$ and $\rho = 9$, with μ=1 and samples/load=200.000 as before)

Note, the different scales in figure 3.57 result from keeping μ=1 constant. The load is increased by increasing the arrival rate, which reduces the mean inter-arrival time $E[T_A] = \frac{1}{\lambda}$. This causes equally reduced inter-departure times $E[T_D] = \frac{1}{\vartheta}$ because $\vartheta = \lambda$ with loss-less systems.

The *M/G/∞* model is commonly used in *queuing network analysis* to model components that cause a stochastic delay (a random holding time). For example, to model the transmission delay cause by end-to-end IP connections when analysing some overlaid control mechanism implemented across end-systems (application servers and clients), where the queueing devices of interest are the message processing instances rather than the message transmission systems.

Square-root stuffing in dimensioning multi server systems

An interesting approach toward the dimensioning of multi-server systems, which is based on the *M/G/∞* model, is the *square-root stuffing rule* [83],

$$\hat{n}_{\alpha}^{M/M/n} = \lceil \rho + c\sqrt{\rho} \rceil \tag{3.103}$$

where ρ is the mean load given in Erlang, and c is a constant factor that results for the intended service property. This simple rule enables us to dimension the number of servers n of an *M/M/n* system, such that a given percentage α of the served customers happens to be served without delay. This is based on the equal progression of the steady state probabilities $p_i \forall_{i \leq n}$ of *M/M/∞* and *M/M/n* systems; see section 3.1.1 and substitute the constant $e^{-\rho}$ by p_0 in equation 3.102 to unveil the similarity.

The constant c is the solution to $\frac{c F_N(c)}{f_N(c)} = \frac{\alpha}{1-\alpha}$, where F_N and f_N are the *cdf* and *pdf* of the standard normal distribution. This approximation via the *normal distribution* is mathematically validated for large ρ only. However, in practice the *square-root stuffing rule* works well even for small values down to ρ=1 *Erlang* (λ=μ) [83]. In case of finite *M/M/n/s* systems this applies likewise if we substitute the offered load λ by the carried load $\vartheta = \lambda(1 - P_b)$.

Further simplifying the rule, we find $c \approx 1$ for the common rule of thumb in business, which states that 80% of the customers need to be fully satisfied in order to compensate the damage caused by the remaining not so satisfied customers. In case the service rate is given, no waiting is the best possible, and thus, qualifies toward a fully satisfactory QoS.

Virtually infinite number of servers for finite populations – *G/G/c/c/c*

If the customer population is bounded, then a system appears to be infinite if it provides at least one server per potential customer, such that at least one server is available whenever a customer arrives. In this case we again find $T_f = T_h$ and $P_b = 0$ as with *M/G/∞*. For clarity we identify such a

system by *M/G/c/c/c*, as already introduced in section 3.2.4. The state transition diagram depicted in figure 3.58 clearly shows the boundedness of this system. Note that state c is not a blocking state

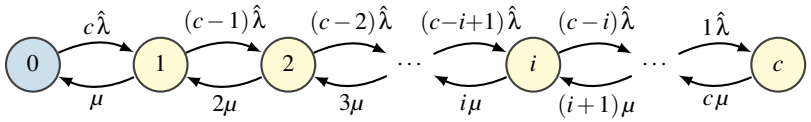


Figure 3.58: *M/M/c/c/c* state transition diagram

because in this state the arrival rate becomes zero. Actually, all *QBD* systems for which at any level the arrival rate, more generally the upward transition probability, becomes zero, are bounded. States from levels above the level with zero upwards transition rate have zero probability in the steady state and do not contribute to the system performance.

Note that for finite populations the offered load is for convenience defined per customer, with $\hat{\lambda}$ *and* $\hat{\rho}=\frac{\hat{\lambda}}{\mu}$ *being the arrival generation rate and a kind of maximum Erlang load per customer*, respectively. These refer to the load that a customer only would generate if all arrivals are rejected. Every accepted arrival reduces this rate because the customer does not generate new arrivals while he is in the system. In consequence is the global arrival process not memoryless. As already sketched in section 3.2.4 published T.O. Engset in [79, 1918] also the steady state probabilities p_i for this system.

$$p_i = p_0 \binom{c}{i} \hat{\rho}^i = \frac{\binom{c}{i} \hat{\rho}^i}{(1+\hat{\rho})^c} \tag{3.104}$$

In the recent literature this model is sometimes referred to as the *Binomial model* because the steady state probabilities p_i are binomially distributed with $\binom{c}{i} \hat{\rho}^i$. Anyhow, being a loss-less, thus blocking-free ($P_b=0$), still finite and queue-less system, makes it quite special.

The effective mean arrival rate per customer, considering the interruption of the arrival generating process by the service interval, can for this system be deduced directly. Being loss-less, the effective arrival rate equals the intended rate $\tilde{\lambda}$. An idle customer generates a next arrival with rate $\hat{\lambda}$, starting instantly after having been served. Once that arrival occurs, in average after one inter-arrival time $t_a=\frac{1}{\hat{\lambda}}$, the customer stops generating new arrivals for the duration of the service time $t_h=\frac{1}{\mu}$. Thus, for every cycle from one service completion to the next, we observe exactly one arrival because no arrivals are ever blocked. The mean length of these cycles is t_a+t_h, and thus is its inverse the arrival rate $\tilde{\lambda}$ we are looking for. Multiplied with the holding time this yields the load generated per customer $\tilde{\rho}$, and further multiplied by the customer population we get the load effectively offered to the system $\rho_s=c\,\tilde{\rho}$. For any loss-less system, the offered load equals the carried load, and due to being in principle a pure loss system, latter equals the mean system filling $\mathrm{E}[X]$. In equation 3.105 these relations are briefly summarised.

$$\hat{\rho} = \frac{\hat{\lambda}}{\mu} \qquad \tilde{\lambda} = \frac{1}{t_a+t_h} = \frac{\hat{\lambda}}{1+\hat{\rho}} \qquad \rho_s = c\,\tilde{\rho} = \frac{c\,\hat{\rho}}{1+\hat{\rho}} = \mathrm{E}[X] \tag{3.105}$$

We notice that the relation between the arrival generation rate $\hat{\lambda}$ and the service rate μ defines a kind of maximum Erlang load $\hat{\rho}$, which is not equal to the load $\tilde{\rho}$ that a customer in average causes. The larger this virtual load $\hat{\rho}$ is, the more is the load effectively offered to the system ρ_s reduced in comparison to models assuming an infinite customer population, where $\rho_s=\frac{\lambda}{\mu}$.

That the effectively offered load equals the system filling results from the fact that for any loss less system the mean throughput ϑ must equal the mean system arrival rate λ_s. Being a queue less multi server system, where each server can serve one client at a time only, we also know that the mean number of busy servers must equal the mean system filling $\mathrm{E}[X]$. And thus, we have $\lambda_s = \vartheta = \mu\,\mathrm{E}[X]$, from which we get the mean system filling $\mathrm{E}[X]$ as stated in equation 3.105. This would as well

result from $\mathrm{E}[X] = \sum ip_i$. However, deducing it not considering states and state transitions, we get a hint on the rather special property of this system: the mean system filling $\mathrm{E}[X]$ is independent of both, the arrival and the service distribution, as approved in figure 3.59. In consequence are the equations stated in equation 3.105 generally *applicable for any G/G/c/c/c system with finite mean arrival and service rates,* $\hat{\lambda}$ *and* μ.

As already done with the unbound *G/G/*∞ system we evaluate the system filling $\mathrm{E}[X]$ and the variation coefficient of inter-departure times c_D for different combinations of arrival and service time distributions (figure 3.59). As expressed in equation 3.105 does the system filling $\mathrm{E}[X]$ not rise linear

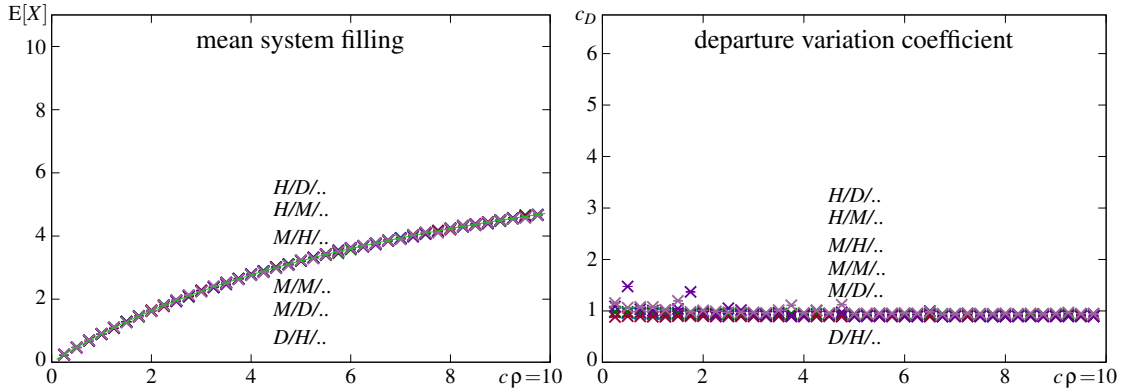


Figure 3.59: Mean system filling $\mathrm{E}[X]$ and departure coefficient of variation c_D of *G/G/9/9/9* (M=Markovian(c_v=1), D=deterministic(c_v=0), H=hyper-exp.(c_v=5), μ=1, samples/point=200.000)

with the virtual system load $c\,\hat{\rho}$, and is independent of both, the service and the arrival distribution. The simulation results fit perfectly to the curve calculated using equation 3.105. Would we rescale the x-axis to reflect the effectively offered load ρ_s, we would again see a linear increase. Without losses all arrivals are served, and thus is the mean number in the system $\mathrm{E}[X]$ equal the effective arrival rate λ_s times the mean holding time t_h (Little's law $N = \lambda T$).

The results showing the departure coefficient of variation c_D in figure 3.59 comprise a bigger surprise. The departure coefficient of variation is seemingly also independent of the arrival and service process. Only at very low system loads we notice that smooth arrivals cause slightly lower variation and bursty arrivals slightly higher. Besides some outliers, which occur with hyper-exponential arrivals only, we get a departure coefficient of variation $c_D \approx 1$. Three effects contribute to this: first, the fact that with finite sources the arrival process is defined per source and these are assumed independent, causing the ingress load to be the aggregate of many independent processes. Accordingly we get for the superposition a distribution that approaches c_v=1 for $c \to \infty$. The second effect is the interruption of the arrival process by the flow process, and the third is the presence of multiple independent server instances.

Figure 3.60 shows the inter-departure time histograms at different system loads. Even at a rather low load of $c\,\hat{\rho}$=1, note that there are 9 servers to handle this ($\rho_s = \frac{c\hat{\rho}}{1+\hat{\rho}}$=10%), we see quite equal departure distributions for all process pairings. Only a slight divergence is visible at T_d=1, which results from deterministic service time t_h=1 in case of *M/D/9/9/9* and *H/D/9/9/9*.

At 50% load (right graph in figure 3.60) we observe that the departure variation is quite independent of the process selection. We again conclude that this results from the many independent servers working in parallel. To see any differences we choose a logarithmic scale on the y-axis and added some more process pairings, particularly *H/H/..* , *E/E/..* , and *U/D/..* . Down to probability density $\mathrm{P}[T_d] < 0.05$ the departure distributions are rather indistinguishable. Below they splits, showing a rather heavy tail for the bursty service process, except if combined with deterministic arrivals. With smooth service processes the tails are below that of *M/M/..* . A divergence at T_d=1 for deterministic serving is at this load level no more discernible.

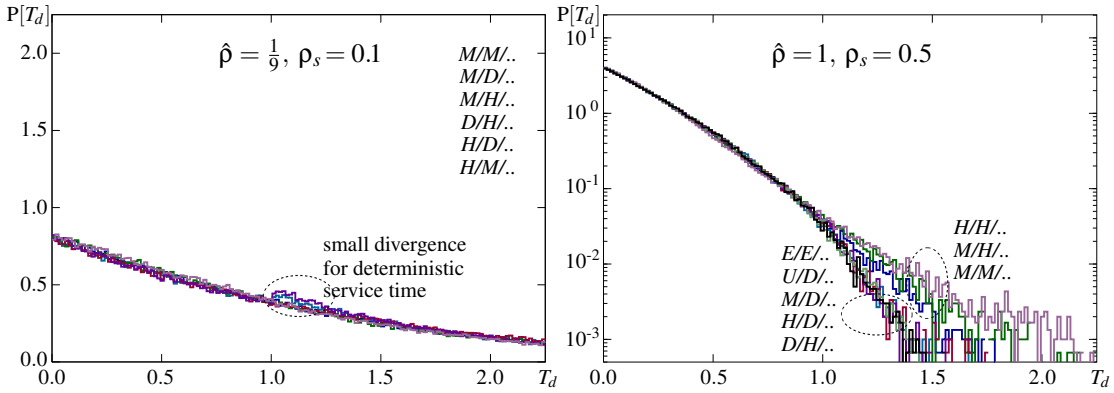


Figure 3.60: Inter-departure time histograms $P[T_d]$ for the examples shown in figure 3.59, $c=9$ (M=Markovian, D=deterministic, E=Erlang^5, U=Uniform$(0,2\lambda)$, H=hyper-exp.($c_v=5$), $\mu=1$, 200k samples)

3.3.3 Processor sharing – *M/G/PS*

In case of *processor sharing*, where the server serves all present customers in parallel, the experienced service time increases the more customers are currently served. The momentary, per customer assigned service share $\sigma(t)$ depends on the current system filling $X(t)$, $\sigma(t)=\frac{1}{X(t)}$. Consequently, the achieved service rate $\sigma(t)\mu$ is not constant and may even change while a customer is served. However, in any state $i(t)=X(t)$ there are exactly i clients served in parallel, such that the server always contributes its full capacity, $i(t)\cdot\sigma(t)=1\,\forall_{i>0}$, as shown with the *M/M/PS* system's state transition diagram, depicted in figure 3.61. The state transition diagram is identical with that for

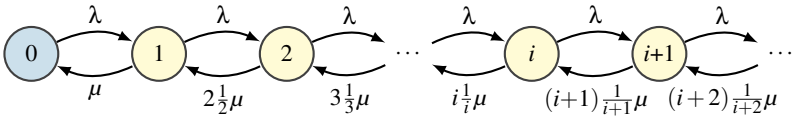


Figure 3.61: *M/M/ps* state transition diagram

M/M/1, and so are the steady state probabilities p_i and all the thereon based performance metrics.

$$p_i = \rho^i(1-\rho) \qquad \mathrm{E}[X] = \frac{\rho}{1-\rho} \qquad \mathrm{E}[T_f] = \frac{1}{\mu(1-\rho)} \tag{3.106}$$

That all states are serving states makes no difference for the continuous time Markov chain analysis, and here the infinite serving states do not cause impractical infinite capacity. In this sense processor sharing represent an infinite server systems, which partly explains the *insensitivity* concerning the service process [83, 84]. Actually, this is a very interesting feature of the *M/M/PS* model because it allows us to *use the results in equation 3.106 for any M/G/PS system.*

For an exemplary proof of the *service time insensitivity* see for example [83, section 22.4], where it is based on Coxian service time distribution and solving the balance equations thereof. Please note that for perfect processor sharing (PS) with zero overhead, it makes no difference how many server units are employed to provide the system capacity μ. Therefore, in theory there exists no multi-server case; and thus, we replace the number of servers n in the Kendall notation by *"PS"* to highlight this peculiarity. In literature the systems are commonly identified as *M/G/1/PS*.

If we restrict the sharing to practically feasible mechanisms we get a plethora of sharing options; way too many and too specific to discuss them in this general context. Please refer to the rich literature on different server assignment schemes, also known as load and/or server scheduling in that context, for models specifically applicable with an implemented sharing mechanism. Here, for this section, we assume the traffic to be like a fluid, *infinitely dividable*, and multiple servers to *jointly perform like a single unit*.

A generally quite important property of *processor sharing* systems is that the *head of the line* blocking cannot occur. With regular queueing systems all jobs in behind a very long one need to wait until that long job is finished before they can proceed in the queue. With processor sharing all the present jobs are served in parallel, and the long job only occupies its share of the system capacity, which only fractionally increases the flow time of the other jobs that happen to arrive while the very long job is processed.

Mean flow time discrimination – egalitarian processor sharing

As usual we call the time span in between the arrival and departure of a job the flow time T_f, and the time the server needs to finish a job on its own, when no other jobs are served in parallel, the holding time T_h. Note that for consistency the term 'holding' is here irritatingly used – actually the server holds a job for its entire flow time. Furthermore, we call the difference among these two the *waiting time*, $T_w = T_f - T_h$, knowing that no queue and thus no waiting exists. This naming convention results from approximating the processor sharing system by time sharing with infinitesimally short time slices $\Delta t \to 0$. With time sharing every job returns to the queue after if received service for the time Δt, until it is finished, when $\sum_i \Delta t \geq T_h(i)$.

This approximation is also used by L. Kleinrock in [85, chapter 4] when deriving the flow time distribution of different scheduling systems. For plain *processor sharing* the conditional mean flow time $t_f(t_h)$ and conditional mean waiting time $t_w(t_h)$ are:

$$t_f(t_h) = t_h \frac{1}{1-\rho} \qquad t_w(t_h) = t_h \frac{\rho}{1-\rho} \tag{3.107}$$

This result reviles that the *mean flow time* and the *mean waiting time* of processor sharing increases *linearly* with the *holding time*, being the effort/load that a job demands/causes. If we assume the flow time to be the primary performance measure, than an *M/G/PS* system *discriminates* longer jobs, at least compared to a common FIFO queueing systems. However, if we calculate the penalty function

$$\frac{t_w(t_h)}{t_h} = \frac{\rho}{1-\rho}$$

we recognise that *the penalty is solely system load dependent*, and thus, *M/G/PS is perfectly fair* because the penalty any job experiences does not depend on its size. A job twice as long as some other experiences also twice the flow time of the other, no more no less. From this we may conclude that *M/G/1/FIFO* systems unfairly privilege long jobs due to the *head of line* problem. The unconditional mean waiting time $t_w = \int t_w(t) f_h(t)\,\mathrm{d}t = \frac{\rho}{\mu(1-\rho)}$ is the same as for *M/M/1*, as is the mean flow time t_f. Only the variance and potential higher moments of the flow and waiting time distributions T_f, T_w are likely worse.

Quite interestingly shows the preemptive, work conserving *M/G/1/LIFO* system the precisely same conditional flow time, and also the service process insensitivity property. This can be roughly explained if we mimic the *M/G/1/LIFO* system by a time sharing system where after some Δt the currently served job returns to the head of the queue, from where it is picked for service again and again until it is finished. For $\Delta t \to 0$ the queueing discipline makes no difference, and thus this equals the approximation used above to derive the conditional mean flow and waiting times for *M/G/PS*.

In addition thereto assert the authors of [86] equivalence between the flow time distribution of processor sharing $F_{T_f}^{G/M/PS}$ and the waiting time distribution of random queueing $F_{T_w}^{G/M/1/RAND}$. Besides a scaling factor, being the probability for arrivals entering a non-empty *G/M/1/RAND* system, the two distributions are probabilistically argued to be identical. However, a complementary evaluation of the theoretic result is not provided.

In figure 3.62 we show simulation results for the mean system filling $\mathrm{E}[X]$ and flow time $\mathrm{E}[T_f]$ for different arrival processes and holding time distributions, together with the analytic results gained

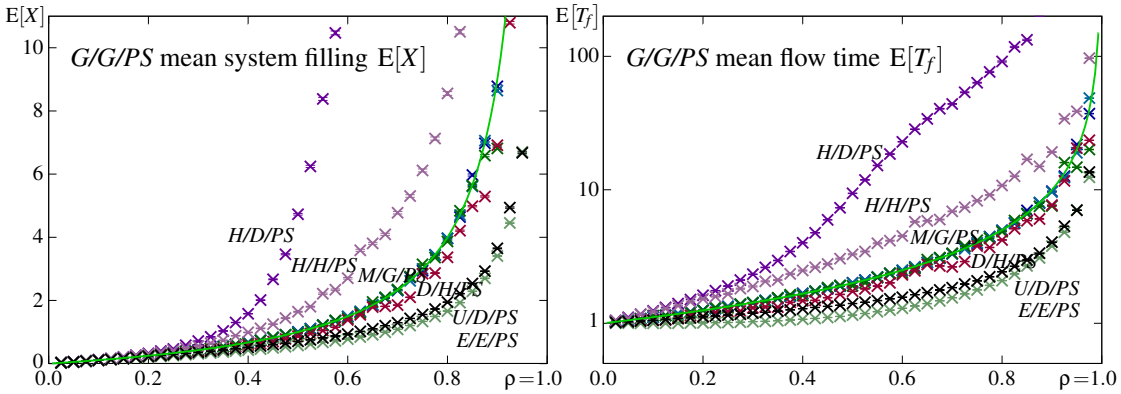


Figure 3.62: Mean system filling $\mathrm{E}[X]$ and flow time $\mathrm{E}[T_f]$ of *G/G/PS* systems
(*M*=Markovian, *D*=deterministic, *E*=Erlang^5, *U*=Uniform$(0,2\lambda)$, *H*=hyper-exp.(c_v=5), μ=1, 200k samples)

from equation 3.106 and equation 3.107. The simulation results for the three cases with Markovian arrivals, being *M/M/PS*, *M/D/PS*, and *M/H/PS*, summarised in the figure as *M/G/PS*, fit perfectly the analytically calculated curve. Only for non Markovian arrival distributions we get diverging results. As can be expected, for bursty arrival distribution the performance is worse, for smooth arrivals it is better. Particularly poor is the combination of bursty arrivals with deterministic service. In this case the processing of a present burst cannot speed up toward its end due to some shorter than average holding times of clients within the burst. This is also visible if we look at the inter-departure time's

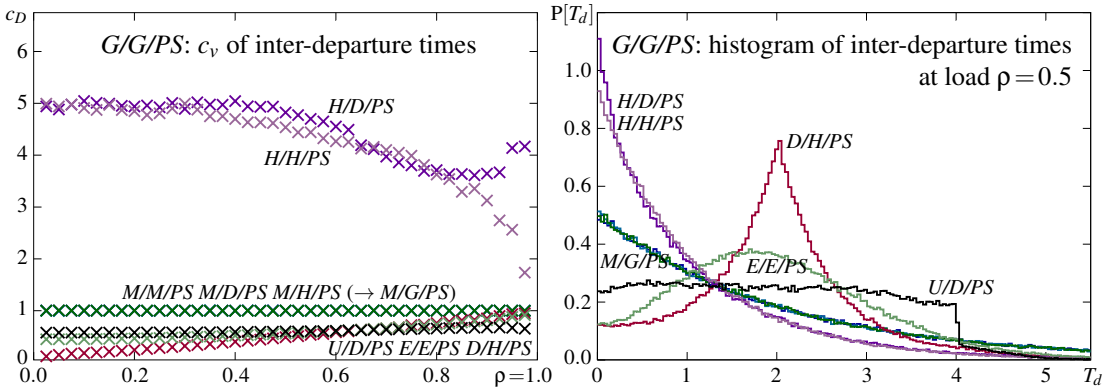


Figure 3.63: Departure variation coefficient c_D and inter-departure time histogram of *G/G/PS* simulated for the same arrival/departure pairings as in figure 3.62

coefficient of variation shown in the left of figure 3.63, where except for deterministic serving we again find $c_D \to 1$ for $\rho \to 1$.

As before with infinite number of servers, a deterministic service process preserves the arrival characteristic, such that it is well discernible in the histogram of the inter-departure times. From our examples we see this in the right graph of figure 3.63 for *M/D/PS*, *H/D/PS* and *U/D/PS*. The peak from deterministic arrivals, shown for *D/H/PS*, as well as the characteristics of other arrival processes vanish with increasing load the more quickly, the more variable the service process is. That the peak occurs at T_d=2, and not at T_d=1 as in figure 3.57 results from the lower load ρ=0.5 used here because at ρ=1 this system becomes instable as its capacity limit would be reached.

Processor sharing is not order preserving. Thus, the inter-departure time, which by definition must be positive, partly results from late departures of clients with long holding times that have arrived prior clients that departed earlier, meaning after clients that overtook them in the time domain while they are served. The higher the system load is, the more often this occurs, and this partly contributes to equalise the arrival variance. In addition, the *momentary service rate varies* during the

service of a client, and thus the effective holding time is a random process even for deterministic service times. This explains the c_D-drop for *H/D/PS* and slight c_D-rise for *U/D/PS*, where overtaking cannot occur. Latter applies as well for *M/D/PS*, though here we have $c_D{=}c_A{=}1$ straight away. However, this randomisation also explains the tail above $T_d{=}4$ in the inter-departure time histogram of *U/D/PS*.

Networks of *M/G/PS* systems

As proven by F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios in [84, 1975] yield networks of *M/G/PS* systems product form results,

$$\mathrm{P}[i_1, i_2, .. , i_j, .. , i_m] = \prod_{j=1}^{m} \mathrm{P}[X_j(t){=}i_j] = \prod_{j=1}^{m} \rho_j^{i_j}(1-\rho_j)$$

where i_j and $\rho_j{=}\frac{\lambda_j}{\mu_j}$ are the current system filling and the mean local load at the involved *M/G/PS* systems j, respectively. The *product form property* enables us to calculate the end-to-end flow time across a chain of queueing systems. Please refer to the literature on BCMP networks for more details.

Discriminatory processor sharing – *M/G/PS/PRIO*

Even distribution of the server capacity to all present jobs is maximally fair. However, jobs are commonly not evenly important/demanding. To assign more capacity to a more important job, and less to a less demanding one, we introduce a set of priority classes $P = \{p_1, p_2, .. \}$. The priority of each class is expressed by a weighting factor g_p. This is used to distribute the serving capacity such that the momentary serving share σ_p, being the percentage that a job of class p gets, is distributed according to the weights for all currently present jobs. The serving share σ_p per job results as

$$\sigma_p(t) = \frac{g_p}{\sum\limits_{P} g_i x_i(t)} \tag{3.108}$$

where $x_i(t)$ is the number of currently present jobs from priority class p_i. The momentary service rate at which a job is processed is $\mu_p(t) = \sigma_p(t)\,\mu_p$. Anyway, note that $\frac{1}{\mu_p}$ is the mean service time of jobs from class p, meaning the time the server in average needs to finish such a job when no other jobs are processed in parallel. Not to be mistaken with $\sigma_p(t)$, which is the achieved service share at some time t, where other jobs may be present, such that $\frac{1}{\sigma_p(t)\mu_p} \geq \frac{1}{\mu_p}\,\forall_t$. We note that $\sigma_p(t)$ is not a constant. It depends on which and how many jobs are currently in service, and may thus change while a job is processed. Actually, no service guarantees are assured, only relational service differentiation is ensured.

Today this priority based sharing scheme is commonly called *discriminatory processor sharing* (DPS), in contrast to the *egalitarian processor sharing* (PS) where no differentiation among flows is foreseen, and the *generalised processor sharing* (GPS), where either the sharing is defined by some positive function based on the current total server population [87], or a guaranteed, load independent share per class and aliquot distribution of not used shares to the classes currently present [88, 89]. Note that the two GPS definitions are mutually contradictory. See for example [90, 91] for reasonably recent surveys on the processor sharing topic.

In 1967 L. Kleinrock published in [92] an analytic solution for DPS, which he called *priority processor sharing*, and compared it with round robin time-sharing (finite constant slot size) and common *M/M/1*/FIFO queueing. The derivation is based on discrete queueing with round robin serving, where the service unit Q (*quantum*), which a client achieves per visit, defines the fixed clock interval. Via limit evaluation $Q \to 0$ he found

$$x_p = \frac{\rho_p}{1-\rho}\Big(1 + \sum_{P} \frac{g_i - g_p}{g_p}\,\rho_i\Big) \quad \text{and} \quad t_f(p,t_h) = t_h\Big(1 + \sum_{P} \frac{g_i}{g_p}\,x_i\Big) = t_h\Big(1 + \frac{1}{1-\rho}\sum_{P} \frac{g_i}{g_p}\,\rho_i\Big)$$

where $x_p = \mathrm{E}[X_p]$, $t_f(p,t_h)$ is the service time t_h conditional mean flow time, $\rho_p = \frac{\lambda_p}{\mu_p}$ and $\rho = \sum_P \rho_p$. However, *these equations are faulty* [93]. Based on a correction done by O'Donovan, who first also published an incorrect solution equalling the results published by Keinrock, the correct results for DPS, briefly sketched shortly based on the summary published in [91], was originally presented by Fayolle, Mitrani, and Iasnogorodski in [93].

To get the correct solution it is necessary to solve the following system of integro-differential equations:

$$\begin{aligned} t_f'(p,t_h) &= 1 + \sum_P \frac{g_i}{g_p} x_i \\ &= 1 + \sum_P \int_0^\infty \frac{g_i}{g_p} \lambda_i \left(1 - F_{T_h(i)}\left(\tau + \frac{g_i}{g_p} t_h\right)\right) \mathrm{d}\tau \\ &\quad + \int_0^{t_h} t_f'(p,\tau) \sum_P \frac{g_i}{g_p} \lambda_i \left(1 - F_{T_h(i)}\left(\frac{g_i}{g_p}(t_h - \tau)\right)\right) \mathrm{d}\tau \end{aligned}$$

Assuming $\sum \rho_i < 1$ this system of integro-differential equations has a unique solution

$$t_f(p,t_h) = g_p \int_0^{\frac{t_h}{g_p}} a(\tau)\,\mathrm{d}\tau + \int_0^{\frac{t_h}{g_p}} b(\tau)\,\mathrm{d}\tau$$

where $a(\tau)$ is the unique solution of the defective renewal equation

$$a(\tau) = 1 + \int_0^\tau a(t)\,\Psi(\tau - t)\,\mathrm{d}t \qquad \text{with} \qquad \Psi(\tau) = \sum_P g_i \lambda_i \left(1 - F_{T_h(i)}(g_i \tau)\right)$$

and $b(\tau)$ results from

$$b(\tau) = \sum_P g_i^2 \lambda_i \int_0^\infty a(t) \left(1 - F_{T_h(i)}\left(g_i(\tau + t)\right)\right) \mathrm{d}t + \int_0^\infty b(t)\,\Psi(\tau + t)\,\mathrm{d}t + \int_0^\tau b(t)\,\Psi(\tau - t)\,\mathrm{d}t$$

Solving this solution exceeds the skills of many engineers and to our best knowledge it has been solved for certain limiting regimes (time-scale decomposition, overload, etc.) and few service time distributions (e.g., deterministic and negative exponentially distributed) only. However, based on the above stated solution some properties and asymptotic bounds of DPS have been proven analytically. Among them the important property that for every possible system state the conditional flow time of some higher prioritised (weighted) class never falls below that of a less weighted class

$$t_f(p_i,t_h) \leq t_f(p_j,t_h) \;\forall_{g_i > g_j} \tag{3.109}$$

and that the mean system filling x_p is upper bounded by

$$x_p \leq \frac{\rho_p}{1-\rho}\left(1 + \frac{\sum_P g_i \rho_i}{g_p(1-\rho)}\right) \tag{3.110}$$

both independent of the service time distribution [91, theorem 2 and 6]. In addition, we notice that for discriminatory processor sharing the conditional flow time is not insensitive to the service time distribution. Here $t_f(p_i,t_h)$ does depend on F_{T_h}, in contrast to egalitarian PS. Thus, the composition of the load matters, particularly if different classes contribute differently distributed service times. The flow time penalty of any class depends on the holding time distribution of all classes. Still,

relative prioritisation and *no starvation* are overall granted by equation 3.109 and 3.110, independent of the holding time composition.

The solution for *M/M/DPS* is also presented in [91,93] and here sketched as well:

$$t_f(p,t_h)_{M/M/DPS} = \frac{t_h}{1-\rho} + \sum_{i=1}^{m} \frac{g_p c_i \alpha_i + d_i}{\alpha_i^2}\left(1 - e^{-\alpha_i \frac{t_h}{g_p}}\right)$$

where m is the length of the vector containing all different $g_i\mu_i$-products, $-\alpha_i$ are the m distinct negative roots of

$$\sum_P \frac{g_i \lambda_i}{g_i \mu_i + s} = 1$$

and c_i, d_i are given by:

$$c_i = \frac{\prod_{j=1}^{m}(g_j\mu_j - \alpha_i)}{-\alpha_i \prod_{j=1|j\neq i}^{m}(\alpha_j - \alpha_i)} \qquad d_i = \frac{\prod_{j=1}^{m}(g_j^2\mu_j^2 - \alpha_i^2)}{\prod_{j=1|j\neq i}^{m}(\alpha_j^2 - \alpha_i^2)} \sum_{j=1}^{|P|} \frac{g_j^2 \lambda_j}{g_j^2\mu_j^2 - \alpha_i^2}$$

This solution can be solved numerically, although in case roots of higher order occur the components need to be adjusted accordingly. Evidently, quite some calculations are required, in particular finding all the roots may be challenging.

However, to directly achieve the *unconditioned mean flow time* $t_f(p)$ of *M/M/PS/Prio* (DPS) the set of linear equations resulting from $t_f(p) = \int t_f(p,\tau)\mu_p e^{-\mu_p \tau}\,\mathrm{d}\tau = \int t_f'(p,\tau) e^{-\mu_p\tau}\,\mathrm{d}\tau)$ needs to be solved [93, lemma 3].

$$t_f(p)\Big(1 - \sum_P \frac{\lambda_i g_i}{\mu_i g_i + \mu_p g_p}\Big) - \sum_P \frac{\lambda_i g_i t_f(i)}{\mu_i g_i + \mu_p g_p} = \frac{1}{\mu_p}$$

Rewriting this to

$$\Big(1 - \rho_p - \sum_{i\neq p} \frac{\lambda_i g_i}{\mu_i g_i + \mu_p g_p}\Big) t_f(p) \; - \sum_{i\neq p} \frac{\lambda_i g_i}{\mu_i g_i + \mu_p g_p} t_f(i) = t_h(p)$$

we get the matrix equation:

$$A \cdot t_f = t_h \quad \text{with} \quad a_{ij} = \begin{cases} -\dfrac{\lambda_j g_j}{\mu_j g_j + \mu_i g_i} & i \neq j \\ 1 - \rho_i - \displaystyle\sum_{k\neq i} \frac{\lambda_k g_k}{\mu_k g_k + \mu_i g_i} & i = j \end{cases} \tag{3.111}$$

Numerically this set of linear equations is easily solved. The initial equation can, without prove, be rewritten to yield the weights g_i required to achieve an intended mean flow time discrimination at a certain intended system load [93]. For two classes, P=$\{1,2\}$ with g_1, g_2, the closed form result

$$t_f(1) = \frac{1}{\mu_1(1-\rho)}\Big(1 + \frac{\mu_1\rho_2(g_2 - g_1)}{\mu_1 g_1(1-\rho_1) + \mu_2 g_2(1-\rho_2)}\Big)$$

$$t_f(2) = \frac{1}{\mu_2(1-\rho)}\Big(1 + \frac{\mu_2\rho_1(g_1 - g_2)}{\mu_1 g_1(1-\rho_1) + \mu_2 g_2(1-\rho_2)}\Big)$$

is available [91,93], and setting g_1=g_2 we get the mean flow time of egalitarian processor sharing, $t_f(p) = \frac{t_h(p)}{1-\rho}$. These analytic results show how complex the path for a mathematical analysis of

queueing systems can be. And thus, sometimes, simulation can be the more practical approach to empirically evaluate specific DPS settings of interest.

The simulation results shown in figure 3.64 to 3.67 try to cover the general behaviour of DPS. A comprehensive evaluation by simulation is out of reach, given the many options to variate system parameters. In figure 3.64 we show per flow and in total the mean system filling $\mathrm{E}[X_i]$ and the

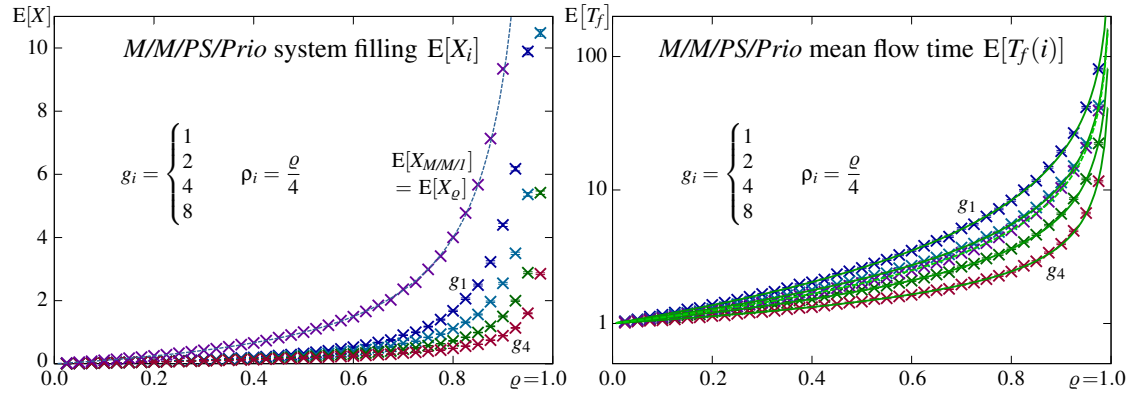


Figure 3.64: Mean system filling $\mathrm{E}[X_i]$ and flow times $\mathrm{E}[T_f(i)]$ of the *M/M/PS/Prio* system
(equal load shares $\rho_i = \frac{\varrho}{|P|}$, increasing system load $\varrho = 0..1$, $\mu = 1$, $200k$ samples per load point)

different mean flow times $\mathrm{E}[T_f(i)]$ per flow i and its average across all flows for negative exponentially distributed holding times and equal load shares per service, together with analytic results. In the left graph we see the different system filling $\mathrm{E}[X_i]$ that results from the different assignment of service capacity only. The total system filling $\sum \mathrm{E}[X_i]$ equals the system filling of the simple *M/M/1* model (dashed line), as we expected given the identical state transition diagram (figure 3.61). Note in this respect that for the total system filling, and the average flow time, the service order is irrelevant. The graph on the right shows the discrimination of the mean flow time $\mathrm{E}[T_f(i)]$. The simulation results perfectly reside on the analytic curves (solid lines), which we get from solving equation 3.111 numerically, point by point.

Next we change the holding time distribution F_{T_h} and analyse how this effects the intended mean flow time discrimination. In figure 3.65 we present the impact of bursty versus smooth high priority traffic, left and right graph, respectively. To keep the aggregate traffic distribution the same, we only

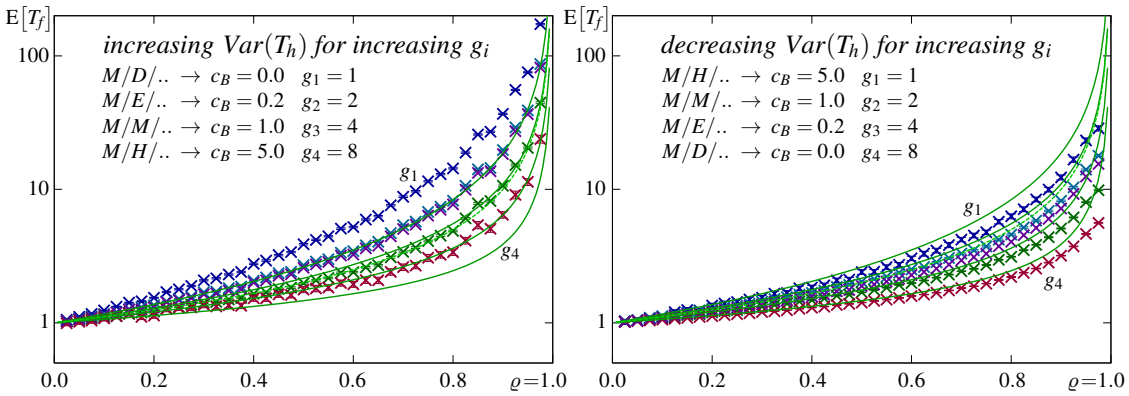


Figure 3.65: Impact of prioritised flow's F_{T_h} on the individual mean flow times $\mathrm{E}[T_f(i)]$
(equal load shares $\rho_i = \frac{\varrho}{|P|}$, increasing system load $\varrho = 0..1$, $\mu = 1$, $200k$ samples per load point)

reverse the order in which we assign the different holding time distributions to flows with increasing priority. The load shares per flow are identical, $\rho_i = \frac{\varrho}{|P|}$, and thus we have the same holding time distribution of the traffic aggregate. Still, the results presented in figure 3.65 differ considerably. The

flow time discrimination for purely Markovian holding times is shown as reference (solid lines). In case of highly prioritised flows with highly varying holding times T_h (left graph) all flows suffer and experience an unsatisfactory increase of their mean flow times. Vice versa, high priority load with smoothly distributed T_h (right graph) improves the performance, such that all flows experience mean flow times that fall below those of *M/M/PS/Prio*, even though the same amount of bursty load is actually present in both cases. This clearly shows that with discrimination the service time insensitivity of egalitarian PS vanishes. However, that all curves rise to infinity not prior approaching overload proves that no starvation occurs, as promised by equation 3.110. For examples with non Markovian arrival distributions please refer to section 4.1.3 on weighted fair queueing (WFQ).

Next we evaluate how different load shares influence the performance. Here, in figure 3.66 and 3.67, we keep the total load constant, $\varrho = 0.8$ and continually increase the most prioritised load while we decrease the least prioritised load by exactly the same amount. In figure 3.66 we evaluate the

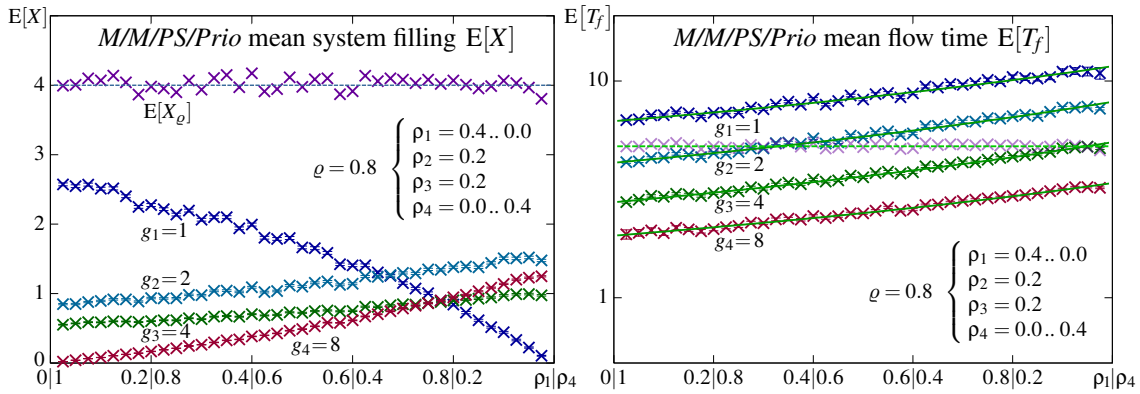


Figure 3.66: Dependence of $\mathrm{E}[X_i]$ and $\mathrm{E}[T_f(i)]$ on load contributions for the *M/M/PS/Prio* system (increasing prioritised share $\rho_{max} + \rho_{min} = 2\frac{\varrho}{|P|}$, system load $\varrho = 0.8$, $\mu = 1$, $200k$ samples per load point)

case of equal Markovian holding time distribution. The mean system filling $\mathrm{E}[X_i]$ is not symmetric because the actual time spent in the server depends on both, the own weight and the shares occupied by other traffic classes. The more higher prioritised load is present, the higher becomes the flow time, as clearly visible in the left graph. Again, the simulation results approve the analytic corves and vice versa. The total system filling and the average flow time across all classes remain constant, in accordance with Little's law $N = \lambda T$.

Finally, we try to test equation 3.109, stating that a higher prioritised load is always served better than a less prioritised. Therefore, we restrict us to two classes and two rather extreme holding time distributions, being deterministic (D) with zero variance, $c_B = 0$, and hyper-exponential (H) with a coefficient of variation $c_B = 5$. The results for opposing weight assignment, where a client from the prioritised class gets twice the service capacity of a less prioritised client are shown side-by-side in figure 3.67. As before we recognise that high prioritised bursty load causes a general degradation of the system performance. It causes higher system filling and in strict consequence worse flow times (Little's law $N = \lambda T$). This cannot be compensated by more prioritisation, in contrary, it worsens the higher bursty flows are prioritised. However, for every evaluated load composition we see better performance for the higher prioritised flow, holding time independently, as promised by equation 3.109.

We also recognise that the simulation results are quite unsteady, even though the same number of samples has been generated as before. This results from the special mixture and the absence of any negative exponentially distributed service times. However, the here not shown simulation results for the departure coefficient of variation reveals a quite stable $c_D = 1$ for all flows and in total. This shows the equalisation caused by the service rate variance introduced by parallel serving and affecting all classes similarly.

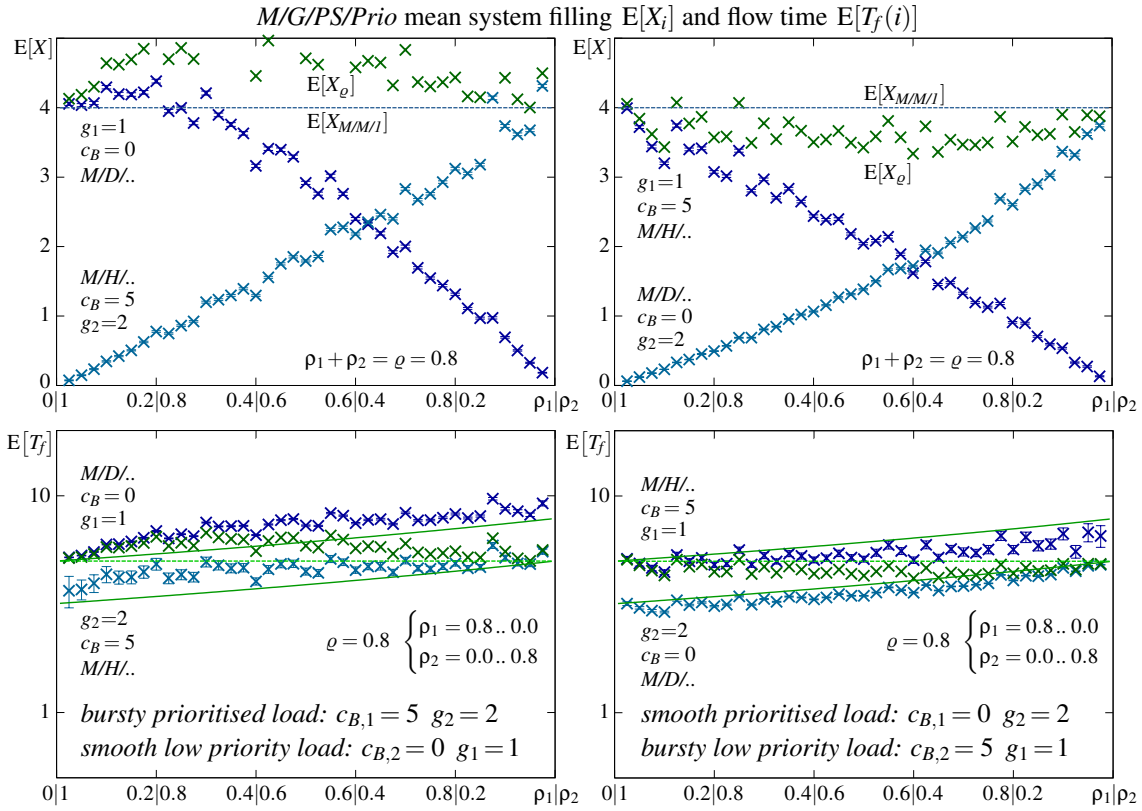


Figure 3.67: Impact of different load composition, in shares and F_{T_h}, on $\mathrm{E}[X_i]$ and $\mathrm{E}[T_f(i)]$
(increasing prioritised share $\rho_2/\rho_1 = 0..1$, system load $\varrho = 0.8$, $\mu = 1$, $200k$ samples per load point)

Finite customer population – *M/G/PS/c*

To complete the discussion of processor sharing systems we finally address the finite customer population case, assuming the Engset setting, where customers do not generate new arrivals while being served. The Markov chain for negative exponentially distributed arrival and service times is shown in figure 3.68. We recognise that the state transition diagram is identical with that for the

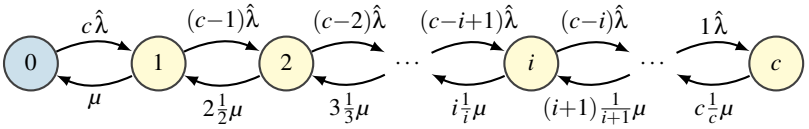


Figure 3.68: *M/M/ps/c/c* state transition diagram

single server queueing system where the system size equals or exceeds the customer population, which we identify as *M/M/1/c/c* model and briefly sketched in section 3.2.4. Thus, we could apply the procedure outlined in section 3.2.4 to get the steady state probabilities p_i and all the thereon based performance metrics. However, instantly applicable analytic results for the state probabilities p_i of the single server *M/M/1/c/c* model are readily presented in [14, section 3.8], and thus here repeated.

$$p_i = \hat{\rho}^i \frac{c!}{(c-i)!}\, p_0 \qquad p_0 = \frac{1}{\sum_{i=0}^{c} \hat{\rho}^i \frac{c!}{(c-i)!}} \tag{3.112}$$

We notice that this system is virtually infinite, there exists no blocking state, and thus it is loss-less. Accordingly equals the throughput the offered load, $\vartheta = \lambda_s$. However, due to the varying

holding times caused by the resource sharing, the effectively offered load diverges from the indented load, $\lambda_s \leq c\tilde{\lambda} = \frac{c\hat{\lambda}}{1+\tilde{\rho}}$, in contrast to the also loss-less virtually infinite multi server case *M/M/c/c/c* presented in the end of section 3.3.2.

Setting up the λ_s calculation via the state transition diagram shown in figure 3.68, we find the quite evident relation with the mean system filling, $\lambda_s = \hat{\lambda}(c - \mathrm{E}[X])$.

$$\lambda_s = \sum_{i=0}^{c} p_i(c-i)\hat{\lambda} = \hat{\lambda}\Big(\sum_{i=0}^{c} c\,p_i - \sum_{i=0}^{c} i\,p_i\Big) = \hat{\lambda}\left(c - \mathrm{E}[X]\right) \tag{3.113}$$

A comparison of the *M/M/PS/c* model with the *M/M/1/c/c/LIFO* model is here skipped. The considerations based on the time-sharing mechanism sketched in [14] for the Erlang setting can probably be applied in a similar way.

In figure 3.69 we show simulation results for the mean system filling $\mathrm{E}[X]$ and flow time $\mathrm{E}[T_f]$ for different arrival processes and holding time distributions, together with the analytic results gained via calculating all system states using equation 3.112. The simulation results for all process combinations

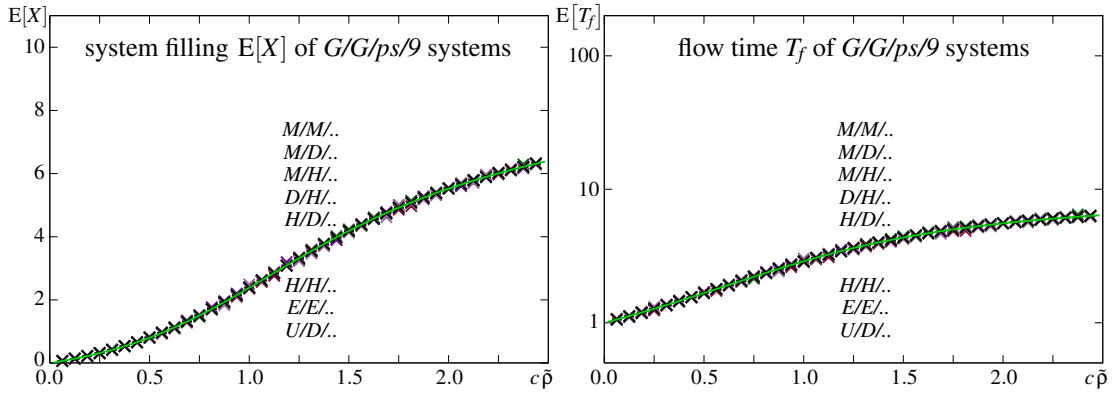


Figure 3.69: Mean system filling $\mathrm{E}[X]$ and flow time $\mathrm{E}[T_f]$ of *G/G/PS/c* with $c = 9$ customers
(*M*=Markovian, *D*=deterministic, *E*=Erlang^5, *U*=Uniform$(0, 2\lambda)$, *H*=hyper-exp.(c_v=5), μ=1, 200k samples)

studied fit perfectly on the analytically calculated curves. This again shows the influence of the arrival process interruption while a customer is in the system. Actually, this system is a little ill conditioned because the longer it takes to serve a customer, the less load is generated. In case of processor sharing this causes an effective ingress load ρ_s below the intended load $c\tilde{\rho}$ over which we show the results here.

Generally, for processor sharing with Engset setting we get $\lambda_s \leq c\tilde{\lambda} \leq c\hat{\lambda}$ due to the in average extended time customers remain in the system. Overload is thus also not possible with these systems: if the flow time approaches infinity because of too many customers being served in parallel, the effective ingress rate drops to zero. This has a stabilising effect in this setting, which might contribute to the rather complete process insensitivity found here and also shown in figure 3.70. Besides some outliers, mostly for the effective arrival rate, the coefficient of variation for both, the arrival and the departure process is very close to c_X=1. Note that due to the interruption of the arrival process the *effective* arrival coefficient of variation is not that of the process we specify with the Engset setting to model the arrival generation, which here starts at service completion (the departure event) and not at the latest arrival event.

Also the histogram of the inter-departure times shows no remains of the processes involved. The dependence of the effective arrival process on the service process and the current system state eliminates them quite effectively. In consequence, we may assume that the analytic calculation of the state probabilities p_i stated in equation 3.112, which actually was derived for *M/M/ps/c*, is *at least approximately applicable for many process pairings*.

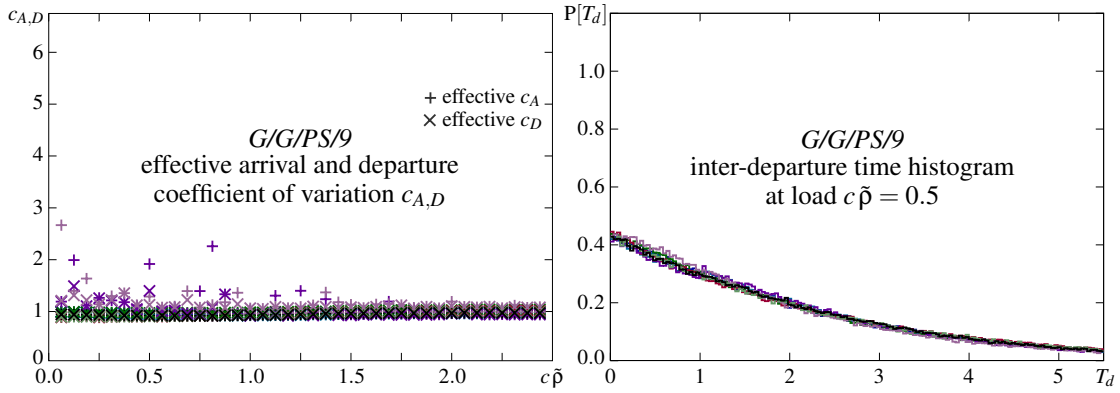


Figure 3.70: Effective arrival and departure coefficient of variation $c_{A,D}$ and inter-departure time histogram of *G/G/PS/c* with $c = 9$ simulated for the same arrival/departure pairings as in figure 3.69

Although finite, this system is again *insensitive* to the holding time distribution. This property we commonly find when serving starts immediately with the arrival to the system, independent of how the processing proceeds [83, page 257]. For example, repeatedly interrupted, served at a constant rate, or with a dynamically slowed down rate. In consequence, this virtually infinite system should yield *product form* results as well. That the *M/M/ps/c* model seems to be insensitive concerning the arrival generation process as well, is here not proven analytically. That the simulation results fit the analytic curves for all the eight combinations evaluated, could be by chance or may have resulted from a faulty implementation – neither reason can be entirely excluded with finite simulation effort.

Processor sharing variants

Generally, the *processor sharing paradigm* presumes the traffic to be like a fluid, infinitely dividable. This represents an optimum that practical resource sharing may intend to approach. Some common scheduling policies come quite close; for example *weighted fair queuing* (WFQ), which preforms the sharing packet-by-packet. See section 4.1.3 for details on WFQ and related policies.

In practice the processor sharing mechanism may be upper bounded, such that only a limited number of customers can become served in parallel [94, 95]. This upper binds the penalty on the effective holding time, but in consequence this introduces blocking if an insufficiently huge waiting queue is added. The blocking probability that results in case no waiting queue is added, we show as an example in figure 3.71 for the Engset setting. Adding a waiting queue opens up a *plethora of options* for controlled processor sharing. Please refer to the rich literature on *combined queueing and sharing* concepts, among others focusing on parallel computing, process optimisation and production line management. Note that mathematicians prefer the term *sojourn time* over the more technical term *flow time*. Thus, more mathematical literature can be found by adding the keyword *sojourn time*, whereas using the term *flow time* tends to yield the more implementation centric, engineering literature on the same issues. However, less specific more generally used terms apply as well, for example *system delay* and *forwarding latency*.

In figure 3.71 we show simulation results for limited processor sharing with Engset setting, where the number of parallel processed customers is upper bound. This represents a loss system, and therefore we show for different arrival process and holding time pairings the mean system filling $\mathrm{E}[X]$ and flow time $\mathrm{E}[T_f]$ together with the blocking probability P_b. Again we see no differentiation in the gained simulation results among any of the arrival/service process-pairings studied. The analytic curves included (dash-dotted) are those found without the bound, using equation 3.112 as before in figure 3.69. Comparing these with the simulation results we recognise that the unbound system provides zero blocking at the price of increased mean flow times. This is rather evident. However, at low intended loads $c\tilde{\rho} < 1.0$ the time advantage is negligible, whereas the blocking remains above

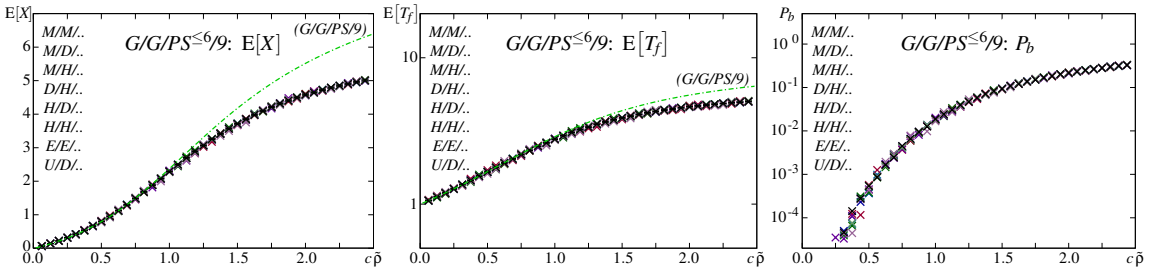


Figure 3.71: Mean system filling $\mathrm{E}[X]$, flow time $\mathrm{E}[T_f]$, and blocking probability P_b of *G/G/PS*$^{\leq 6}$*/9* (*M*=Markovian, *D*=deterministic, *E*=Erlang^5, *U*=Uniform$(0,2\lambda)$, *H*=hyper-exp.($c_v=5$), $\mu=1$, 200k samples)

acceptable rates of $<10^{-3}$ down till $c\tilde{\rho}<0.5$. Thus, a bound on the number of customers that may share a processor is on its own rather useless, at least in terms of system performance.

Another extension of processor sharing results if we assume several servers, each performing processor sharing. Here the core topics are *load balancing* strategies and the overhead associated with the *migration of jobs* among servers. The target is optimally efficient joint performance. A multi-core processor, computer clusters, cloud services, and distributed data centres are among other less data processing centric systems, typical examples thereof.

Parallel network trunks jointly shared by individual flows comprise a similar situation. With label switching the migration is rather free of overhead and in case of packet-switched parallel trunks these can be virtually merged into a single trunk with minor implications (on a packet-by-packet basis). For lower layer date transport this is commonly not that easily possible. For example, heterogeneous wireless radio cells where different radio and base-station technologies regionally overlap each other (e.g., Bluetooth, WLAN, or UMTS-LTE). In this case seamless switching from one radio interface to some other is commonly not possible without considerable extra effort. The mandatory *make before break* strategy, meaning to establish the new connection prior tearing down the existing one, may cause prohibitive overhead when the shared transmission capacities are scarce.

Finally, the insensitivity in respect to the holding time distribution $F_{T_h}(t)$ renders *egalitarian processor sharing* models particularly appealing for problems where the service process characteristics are unpredictable, persistently changing, or too complex to be modelled in detail. For example, *M/G/PS* can be used to approximately analyse multiplexing systems, even if the aggregate flow is composed of an unpredictable mixture of different holding time characteristics, representing different applications' load contribution each. However, the inter-arrivals times need to be Markovian, and the service sharing process per se needs to be time invariant, such that at any time a customer always gets the same share $\sigma_i(t)$ if also the current system filling $X(t)$ is the same. Latter does per se not exclude prioritisation, we could extend equal system filling to equal momentary load composition, meaning equal $X_p(t)\ \forall_p$. However, prioritisation eliminates the holding time insensitivity, and therefore, as of today, systems based *discriminatory processor sharing* need to be analysed case-by-case.

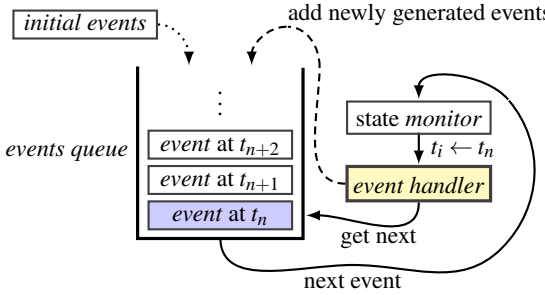
We also should not forget that all the processor sharing models presented in this section yield rather optimistic performance predictions compared to what real resource sharing systems can achieve in practice, where the load is commonly not infinitely dividable. For dimensioning purposes these models are thus not the safest choice, but serve very well as target benchmark when it comes to the optimisation of practical resource sharing mechanisms.

3.4 Queueing system simulation

In the preceding sections we presented the analytic treatment of queueing systems and proved the presented characteristics by simulation results. We did so because this is the simplest way to verify analytic derivations. However, these result are not new, they all can be found in the relevant literature in the presented or some related form, and thus, they do not require any further approval. Vice versa, we included simulation results to approve the developed simulation procedure. In this section the used simulation procedure is outlined together with the underlying concepts, step by step, topic by topic. The here introduced scheme will be maintained and extended in the upcoming chapters in order to evaluate the practical traffic engineering approaches and systems discussed. Only the specific extensions will be presented in detail henceforth.

3.4.1 Event based simulation and the elimination of absolute time

Event based simulation is a technique well suited to evaluate processes where changes occur randomly distributed over time rather than continuously. In such an environment it is perfectly sufficient to restrict the treatment of the system to the instants when changes happen. Potential changes we call *events*, noting that not every event treated must inevitably cause a true change, and the time instants at which these happen and are consequently treated, we call *event times*, being the actual system observation instants. It is not necessary to monitor the system in between subsequent observation instants because we can postulate that during this time span the system is, and remains, in the state it entered after the latest event has been handled. The time that passes in between events we call *inter event time*, and note that in case of simultaneous events this can be zero. Figure 3.72 sketches the basic elements and the execution flow.



component	role
events queue	chronological sorted list of events
event	holds *type* and execution *time*
event handler	performs event-type specific tasks
state *monitor*	records whatever is of interest

Figure 3.72: Event based simulation – core components and execution flow

The central element of event based simulation is the *events queue*; it holds all upcoming *events* for which the *event time* is already known in order of their event times. The basic elements are the *events*; these comprise of an *event type* and the time at which they occur. The functional element is the *event handler*; depending on the event type it performs a sequence of predefined tasks, which may or may not result in the addition of new events to the events queue. Whenever handling an event finishes the next event is taken from the events queue. A *monitor* instance is required to record the system state or any other attribute required. Event specific monitors need to be part of the according event handling procedure, whereas general system monitors are best placed as shown. The simulation starts with initialising the events queue with one or more events. This triggers the execution until the events queue becomes empty again, which ultimately terminates the simulation. Actually, as the name already expresses, *time* plays no role for the execution; it is the currently handled event's time tag, nothing more. Only for debugging and normalising it may be required.

For a *G/G/1* queueing system with a single arrival process and a single server the events queue holds at maximum two values only, the *next arrival time* and the *next departure time*, latter only if

the server is currently busy. Therefore, we do not need a sorted queue, we store these two values in the variables *in* and *out*, as shown in algorithm 3.1. Event by event we select the one closer to the

Algorithm 3.1 Basic *G/G/1* simulation core

```
 1: function GG1SIM(a,A,b,B,s_r,s_t)
 2:     in = 0; out = Inf; t_i = t_n = 0; X = 0; X_m = [ ]; simT = 0;        ▷ initialise variables
 3:     i = j = 0; m = false;
 4:     repeat
 5:         if (i = s_t + 1) then m ← true;                                  ▷ start state recording phase
 6:         else if (length(X_m) = s_r) then m ← false;                      ▷ stop if record size is reached
 7:         end if
 8:         t_n ← min(in,out)                                                ▷ get time of next event to handle
 9:         if (m = true) then
10:             X_m ← [X_m,(t_n − t_i)X];          ▷ add time weighted system state X to states record X_m
11:             simT ← simT + t_n − t_i;                                     ▷ sum up simulation time simT
12:         end if
13:         t_i ← t_n;                                                       ▷ set simulation time to next events time
14:         if (in = t_i) then
15:             if (m = true ‖ i ≤ s_t) then                                 ▷ if more events are required
16:                 in ← t_i + feval ('distrGen',A,num2cell(a(1,:)){:});     ▷ generate next arrival time
17:             else in ← Inf;                                               ▷ else stop generating arrivals
18:             end if
19:             X ← X + 1; i ← i + 1;                                        ▷ handle current arrival
20:             if (out = Inf) then                                          ▷ if server is idle
21:                 out ← t_i + feval ('distrGen',B,num2cell(b(1,:)){:});    ▷ generate next departure time
22:             end if
23:         else if (out = t_i) then
24:             X ← X − 1; j ← j + 1;                                        ▷ handle current departure
25:             if (X > 0) then                                              ▷ if customers remain
26:                 out ← t_i + feval ('distrGen',B,num2cell(b(1,:)){:});    ▷ generate next departure time
27:             else out ← Inf;                                              ▷ else not, reset it to infinite = idle
28:             end if
29:         end if
30:     until (in = Inf & out = Inf)          ▷ until no events remain = event queue becomes empty
31:     return X_m/(simT/length(X_m));        ▷ return the time normalised customer in system record
32: end function
```

recently handled event's time t_i in order to determine the next event type (arrival or departure) and the time t_n it will occur. The difference $t_n - t_i$ expresses the duration of time during which nothing happens and thus, the time the system remains in state X. Accordingly, in order to correctly record the system filling, we record the current state X weighted by the time $t_n - t_i$ it persists (line 10). After doing so we step from the current time t_i to the time of the just determined next event (line 13) and handle the event depending on its type; lines 15-22 and 24-28 for arrivals and departures, respectively. In the end (line 31) we normalize the time weighted states recoded in X_m by the average duration ($\frac{simT}{\text{length}(X_m)}$), being the total time that passed while we are monitoring states divided by the number of states recorded. This assures that the returned sample is independent of the actual durations and can be statistically evaluated to get the mean system filling $\mathrm{E}[X]$, the confidence interval thereof and other statistical properties as required.

The state variable X tells the number of customers currently in the system, either in service or waiting. Every time an arrival occurs we increment it by one, and with every departure it is decremented by one (lines 19 and 24 in algorithm 3.1 respectively). Note that in a time continuous regime nothing happens at the absolutely same time instance; only numerically this may happen, causing $t_n - t_i = 0$. Consequently, the intermediate infinitely short state has zero weight and thus the order we handle such virtually simultaneous events has no impact on the result.

Basically the above outlines the entire simulation procedure; however, to keep it running we have to generate a next arrival whenever an arrival event is handled (line 16). Evidently, when an arrival occurs while the server is idle this arrival triggers a new departure event that needs to be generated instantly (line 21). Likewise, every departure triggers a new departure event if there are customers in the system awaiting service (line 26).

To initiate the simulation, the next arrival time *in* is initialised with zero, triggering the first arrival scheduled at time zero. The next departure time *out* is initialised as infinite, expressing that the server is initially idle. The state monitor X is initialised with zero, representing an idle system awaiting the first arrival at time zero. The simulation should not terminate prior the requested state record size s_r is reached. Up front, a transient phase of s_t arrivals is added; during this the queueing system shall equalise any initial conditions while nothing is recorded. If the system state is recorded or not is determined by the monitoring flag m. It is initialised as `false` and set to `true` when the transient phase has passed. Finally, when the requested number of states has been recorded the flag is reset to `false` again, in order to not record the clear-out phase when remaining events are processed. The simulation terminates implicitly; due to stopping the arrival generation (line 17) the events queue becomes empty when all remaining customers were served and have departed, which stops the execution loop (line 30).

Figure 3.73 depicts the assignment of event times t_i. The simulation starts with the first event scheduled at time t_1, being the left most arrival on the time axis. Its occurrence time can be chosen arbitrarily, but commonly it is set to zero as depicted and coded. Some time into the simulation execution, indicated by the interrupted time axis, we index event times relative to the current event that occurs at time t_i for clarity. Actually the indices count from one, assigned to the first event, to whatever number required. Just before handling the i_{th} event we record the current system state X

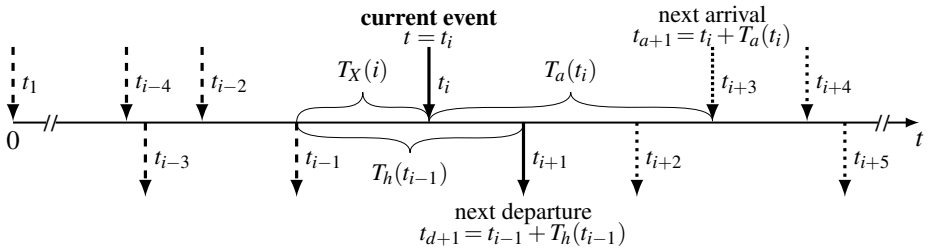


Figure 3.73: Absolute time assignment

weighted by the time it persisted, being $T_X(i) = t_i - t_{i-1}$ (line 10). Than we handle the event that occurs at time t_i, which may generate new events in the future. In case of an *arrival event*, $t_i = t_a$, a next arrival event at time $t_{a+1} = t_i + T_a(t_i)$ is generated and its occurrence time stored, $in \leftarrow t_{a+1}$ (line 16), as long as the simulation has not reached a predefined termination condition (line 15). If the server was idle, meaning that currently no departure is scheduled (line 20), also a departure event scheduled to occur at time $t_{d+1} = t_i + T_h(t_i)$ is generated and stored in $out \leftarrow t_{d+1}$ (line 21). In case the current event is a *departure event*, $t_i = t_d$, only a departure event scheduled at time $t_{d+1} = t_i + T_h(t_i)$ may be generated and stored, $out \leftarrow t_{d+1}$ (line 26), given there are customers still waiting after the current departure happened, meaning $X_{t_{i+}} > 0$ (line 25).

Particularly note that for the events scheduled in the future *the actual event index cannot be determined until it becomes the current event*. Any event that occurs prior a scheduled one, including the currently handled, can generate events that occur prior an already scheduled event; meaning that any time new events may enter the events queue in front of an already scheduled. Therefore, we use the t_a and t_d notation to identify current and future arrivals and departures respectively. Recording the t_d in addition to the system state X enables us to statistically determine the departure process (see section 3.4.2).

While the random numbers $T_X(i)$ result from the difference among the actual timing of subsequent events, the inter-arrival times $T_a(t_i)$ and the service times $T_h(t_i)$ (holding times) result from the rates and distributions (random processes) that define the simulation scenario. The arrival and service process rates a and b as well as their distributions A and B are defined by the parameters handed over to the simulation core. These are nowhere required within the simulation routine, they are passed over to the generation function '`distrGen`', executed via the function evaluation command `feval`. The '`distrGen`' procedure comprises a *switch* to select a particular random number generator based on the string A or B indicating the requested distribution and a *parser* that transforms the distribution parameters array, a or b, into the format that the actual random number generation function requires. Many distributions can be generated by functions natively provided by *Octave*, others have been written on demand. See chapter 2 for details on distributions. If the distributions A and B are a priory given, the according random number generation could be inserted directly, instead of calling '`distrGen`' via `feval`, to save some execution overhead.

In contrast to the basic event indexing introduced above, we count the arrivals and departures separately in i and j, such that the event index actually is $i+j$. We use the individual indices to find out when the transient phase has passed and whether in the end all arrivals also departed ($i=j$). Note that according to common practice the pseudo-code presented does not include parameter checking and other non essential runtime monitoring, including the mentioned check if all arrivals are accompanied by a departure when the simulation terminates, among many other checks alike.

Queueing system optimised time unit

The *time unit* is not relevant for the precise execution of an event based simulation, it is only required for the correct normalisation of the recorded sample. Thus, any time unit is as good as any other. For queueing systems, and communication systems in general if seen from the layer above, the commonly applied *time unit* is the *mean holding time*

$$1\,\textit{time-unit} = \textit{mean-holding-time} = \frac{1}{\mu} \tag{3.114}$$

which is also the basis of the traffic unit *Erlang*

$$1\,\text{Erlang} = \textit{mean-arrival-rate} \times \textit{mean-holding-time} = \frac{\lambda}{\mu} \tag{3.115}$$

traditionally used to express the mean ingress load to the system analysed.

The time unit selection expressed by equation 3.114 implicitly reduces the potential for numeric errors and at the same time makes results comparable straight away, meaning load unit transparent because thereby the utile Erlang unit is implied. If high numeric precision is not required for other reasons, for example vanishingly small state probabilities, this a priori time normalisation reduces the computational effort. In consequence, the time required to run simulation studies may be reduced by this smart time unit choice. Calculators have no means of units, anyhow.

Present time as global time reference

Numeric overflows are another problem that should be systematically avoided because simulation runs can be very long, or a priori unbounded in case of on-line on-demand transient analysis. In particular, the simulation time shall never exceed the numerically representable range. Either we apply a *time reset routine* that catches and mitigates a potential overflow by deducing a certain amount form all time variables once a critical range has been reached, or we avoid these critical numeric areas entirely. Latter is achieved if we eliminate the absolute time issue by referring all time variables to the current instant in time, the *present time*, similar to our daily life where we preferably refer to the past and the future, rather than absolute times.

In order to make this reference shift we need to reformulate some variables and calculations. Figure 3.74 depeicts the new time assignment. Events are now only counted, indexed with i, but have no static assigned occurrence times. The currently handled event e_i defines the time reference

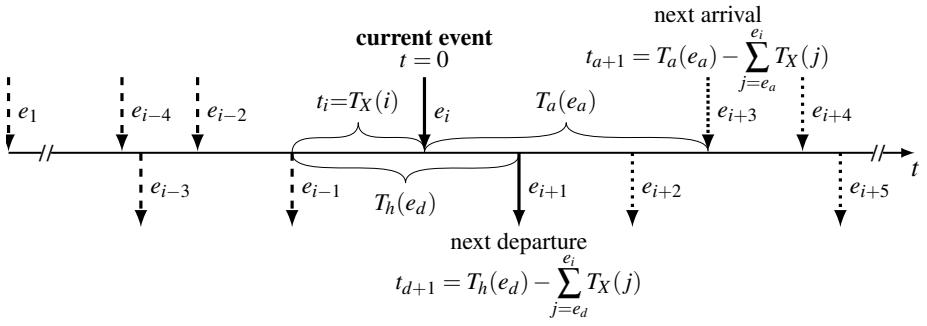


Figure 3.74: Relative time assignment and its maintenance

$t(e_i) = 0$. Based on this instant only three times, precisely durations, are specified and maintained.

$$\begin{aligned} t_i &= \min\left(t_{a+1}, t_{d+1}\right) \\ t_{a+1} &\leftarrow t_{a+1} - t_i \\ t_{d+1} &\leftarrow t_{d+1} - t_i \end{aligned} \tag{3.116}$$

The *state persistence time* $t_i = T_X(i)$ is the duration since the last event has been handled, and it expresses the duration while nothing changed prior the current event. It equals the time that passes in between the last handled and the current event. Accordingly, it needs to be calculated prior the times contained in the *minimum* calculation on the right side are updated as specified in the following two lines. The latter durations t_{a+1} and t_{d+1} specify the current time lag till the next arrival and departure event respectively. Whenever the current time is reset to zero, such that $t(e_i) = 0$ these two need to be decremented by t_i in order to subtract the time that has passed, such that the remaining value equals the remaining time to the event's occurrence.

◊ This *sequential updating has to be performed for all* monitors *likewise* and commonly requires to increment them by t_i in order to reflect the time that has passed since a certain event occurred.

Evidently, the time lag to the current event becomes zero, and this indicates the event type that is to be handled. While handling the current event, new events may be generated and scheduled. The time lag for a newly scheduled event equals the value calculated for the distribution A or B and the rates and parameters contained in a or b, which specify the simulation scenario. At the time generated $t_{a+1} = T_a(e_i) = A(a)$ and $t_{d+1} = T_h(e_i) = B(b)$.

The event by event required *minimum* operation and the two *subtractions* presented in equation 3.116 are rather simple numeric operations, even though real numbered. As only three variables are involved, these reside in registers of the processor once the first is executed, such that no additional data loading from memory is required to perform the other two. The added computational effort is in practice marginal, far less than what one might expect from the complex equations required to express t_{a+1} and t_{d+1} in general, as shown in figure 3.74. In practice these summations contain only one term as the updates are performed sequentially event by event.

Finally we note, the elimination of absolute time is not a necessity for event based simulation; it is a naturally supported feature that provides solid advantages. The computational effort to achieve it is minimal and potentially compensated by efforts saved where relative times are anyhow required.

Changes for *present time based* operation

```
— initialise variables —
...   remove t_n   ...                                   ▷ absolute time no more required: t_n → t_i
— handle any next event —
...   t_i ← min(in, out);                                ▷ replace t_n → t_i
      in ← in − t_i;                                     ▷ decrement time till next arrival
      out ← out − t_i;   ...                             ▷ decrement time till next departure
...   X_m ← [X_m, t_i X];                                ▷ replace (t_n − t_i) → t_i
      simT ← simT + t_i;   ...                           ▷ replace (t_n − t_i) → t_i
...   replace former t_i by eps   ...                    ▷ current time (now) is (numerically) zero
— handle an arrival event —
      if (in ≤ eps)   ...                                ▷ if now an arrival event occurs
...   in|out ← feval(..);   ...
                  ▷ time till new scheduled arrival | departure = inter-arrival | service time T_A(i)|T_B(i)
— handle a departure event —
      else if (out ≤ eps)   ...                          ▷ if now a departure event occurs
...   out ← feval(..);   ...
                  ▷ time till newly scheduled departure = service time T_B(i) (if scheduled)
— normalise and return recorded samples —
...   no changes   ...
```

3.4.2 Single server queueing system evaluation by simulation

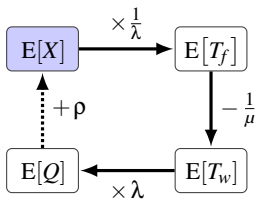
The simulation procedure explained above and basically expressed by algorithm 3.1 returns the system states recorded $X_n = \{x_i\}$. This chronological list of states can be statistically evaluated to derive the *mean number of customers in the system* $\mathrm{E}[X]$, the confidence interval $\mathrm{E}[X] \pm \Delta_X$ thereof, the standard deviation σ_X, as well as higher moments. The *Octave* shell provides commands to directly extract these, i.e.:

$$\mathrm{E}[X] = \frac{1}{n}\sum_{i=1}^{n} x_i = \texttt{mean}(X_m) \qquad \Delta_X = \texttt{confid}(X_m, 95)$$

$$\sigma_X = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i^2 - \mathrm{E}[X]^2\right)} = \sqrt{\mathrm{E}[X^2] - \mathrm{E}[X]^2} = \texttt{std}(X_m) \qquad c_X = \frac{\sigma_X}{\mathrm{E}[X]} = \frac{\texttt{std}(X_m)}{\texttt{mean}(X_m)}$$

Higher moment can be extracted by the $\texttt{moment}(X_n, i)$ commend, where i indicates the moment order. Please refer to the online help of *Octave* to learn more on these commands, and to section 1.5 for the theoretic treatment of statistical properties.

Based on the *mean system filling* $\mathrm{E}[X]$ the other mean characteristics of the queueing system can be calculated. Figure 3.75 shows the sequence in that this is best performed. First we use Little's



$$\mathrm{E}[X] = \texttt{mean}(X_m)$$
$$\mathrm{E}[T_f] = \tfrac{1}{\lambda}\,\mathrm{E}[X]$$
$$\mathrm{E}[T_w] = \mathrm{E}[T_f] - \tfrac{1}{\mu}$$
$$\mathrm{E}[Q] = \lambda\,\mathrm{E}[T_w] = \mathrm{E}[X] - \rho$$

Figure 3.75: Calculating $\mathrm{E}[T_f]$, $\mathrm{E}[T_w]$ and $\mathrm{E}[Q]$ based on $\mathrm{E}[X]$

equation $\mathrm{E}[X] = \lambda\,\mathrm{E}[T_f]$ to get the *mean flow time* $\mathrm{E}[T_f] = \mathrm{E}[X]/\lambda$. From $T_f = T_w + T_s$ we get the *mean waiting time* $\mathrm{E}[T_w] = \mathrm{E}[T_f] - \frac{1}{\mu}$, by simple subtraction of the known *mean serving time* $\mathrm{E}[S = T_h] = \frac{1}{\mu}$ also known as *mean holding time h*. Finally, applying Little's equation for the queueing process alone, we get the *mean queue filling* $\mathrm{E}[Q] = \lambda\,\mathrm{E}[T_w]$.

According to Little's equation is the flow time process T_f the multiplication of the arrival process A with the system filling process X. Thus, if the arrival process A does not depend on the system filling X, we can calculate the *flow time variance* $Var(T_f)$ from the *system filling variance* $Var(X) = \sigma_X^2$, using the rules for independent distributions presented in section 1.5.2:

$$Var(T_f) = \tfrac{1}{\lambda^2} Var(X) + \mathrm{E}[X]^2 Var(A) + Var(A) Var(X) \qquad \text{if } A \perp X$$

If the service process S is independent of the flow time T_f and thus also independent of the system filling X, we can further calculate the variance of the waiting time process T_w as well as that of the queue filling as

$$Var(T_w) = Var(T_f) + Var(T_h) - 2\,Cov(T_f, T_h) \qquad \text{if } T_h \perp T_f$$

$$Var(Q) = \tfrac{1}{\lambda^2} Var(T_w) + \mathrm{E}[T_w]^2 Var(A) + Var(A) Var(T_w) \qquad \text{if } A \perp T_w$$

where $Cov(T_f, T_h)$=0 because independent distributions are in general also uncorrelated.

If the arrival process or service process is not independent of the system filling or queue filling, we cannot use this simple variance calculation. In such cases we have to add according monitors to the simulation procedure in order to get second moment statistics. How and where to add such a monitor is shown next, exemplarily for the *waiting time* T_w.

Departure process variance

We notice that the statistics that can be derived from the system state X do not include information about the *departure process* D. The mean departure rate $\mathrm{E}[D]$ is evidently equal the mean arrival rate, $\mathrm{E}[D] = \mathrm{E}[A]$, given that we have an infinite queueing system that cannot loose any customer that once arrives. However, to tell something about the moments of the departure process using simulations we need to monitor it separately.

While customers are present in the queueing system, the output distribution equals that of the service process. When there are no customers present, the output process is zero. Consequently is the output process an interrupted version of the service process. The interruption probability equals $1-\rho$, and thus also the mean interruption interval is that long. However, the distribution of the interruption interval is unknown. Based on this reasoning we may conclude that the *interrupted service process model* should perfectly represent the output process, if we manage to match the second moment. To get the load dependent second moment for a given arrival and service distribution pair at a certain load, we can use analysis or simple simulation, if the former is too bulky.

Inter-departure time monitor extension

```
1:  — initialise —
2:  ...   D = 0; D_m = [];   ...                ▷ initialise monitor and sample array
3:  — next event —
4:  ...   D ← D + t_i;   ...                    ▷ increment inter-departure time
5:  — arrival —
6:  ...   no change   ...
7:  — departure —
8:  ...   D_m ← [D_m, D];                       ▷ add accumulated inter-departure time to sample
9:        D ← 0;   ...                          ▷ reset inter-departure time monitor
10: — normalise and return —
11: return   ...   D_m   ...                    ▷ return recorded sample
```

The sample D_m required to statistically evaluate the output process needs to contain the *inter-departure times* $T_D(i)$. Therefore a dedicated inter-departure time monitor D and a departure time sample array D_m needs to be added and initialised (line 2). The monitor variable D accumulates the

inter-event times t_i that occur in between two subsequent departures. Every time the simulation jumps to the next event we increment the monitor, $D \leftarrow D + t_i$ (line 4), such that it always tells the time that passed since the latest departure occurred. Every time a departure occurs, we add the accumulated value to the recorded sample, $D_m \leftarrow [D_m, D]$ (line 8), and reset the monitor $D \leftarrow 0$ (line 9). Finally the recorded sample is returned (line 11) for statistical evaluation.

Based on the recorded sample D_m the waiting time statistics can be extracted independent of any other process, and therefore independent of the system filling. The statistical metrics can be extracted as before. The required variance $Var(T_D)$ is than simply the square of the standard deviation revealed by $\texttt{std}(D_m)$, and the *coefficient of variation* c_D results from dividing σ_D by its mean $\mathrm{E}[D]$:

$$Var(T_D) = \sigma_D^2 = \big(\texttt{std}(D_m)\big)^2 \qquad c_D = \frac{\sigma_D}{\mathrm{E}[D]} = \frac{\texttt{std}(D_m)}{\texttt{mean}(D_m)}$$

3.4.3 Different queueing disciplines

Commonly we assume *first-in first-out* (FIFO) queueing. At first glance anything different seems awkward. However, there exist mechanisms that implement *last-in first-out* (LIFO) queueing, for example a procedure stack, or some other discipline that schedules customers according to their needs rather than in order of arrival. Latter is best modelled by *random* (RAND) queueing, for example to predict the waiting time distribution of an emergency handling unit where unpredictable ad hoc decisions govern the scheduling.

Reviewing the simulation procedure we recognise that there is nothing present that indicates a certain queueing discipline. For the evaluation of mean properties this is irrelevant; only the higher order moments of the waiting time are affected. To cover these, dedicated waiting time monitors are essentially required because different queueing disciplines are considered by different customer selection strategies applied when a waiting customer becomes scheduled for departure. To realise FIFO the customer that spent most time in the queue is selected, being the one with the waiting time $\max(W)$. For LIFO it is the customer which spent the least time in the queue, being $\min(W)$, and for RAND selection it is any one, being $W(\mathrm{rand}(\mathrm{length}(W)))$.

Dedicated waiting time monitoring

The *waiting time process* T_w is the most interesting characteristic of infinite queueing systems. Queueing space is assumedly of no concern with infinite systems, and at reasonable loads the mean flow time is predominated by the known mean holding time $\mathrm{E}[T_h] = \frac{1}{\mu}$.

To monitor and statistically evaluate the process we need to record all actually occurring waiting times $T_w(i)$. To do so, we initiate a waiting time monitor variable whenever a client arrives and joins the queue of waiting customers. The simulation realises the queue implicitly, it requires a single counter only, and this is provided by the system state variable X. The number of waiting customers is $X - 1$. If an arriving customer happens to initially enter service, when an idle server is available at its arrival event ($X = 0$), the waiting time to be recorded is zero. These zero records are essentially required, else the waiting process would not be correctly reflected by the recorded sample.

A particular complexity arises because there can be more than one customer waiting at a certain instant of time. Therefore, it is required to set up and maintain a set of waiting time monitors $W = [\,]$ in addition to the sample recorded $W_m = [\,]$. For both we define arrays, which can be dynamically extended and shrunken as required, and are initialised as empty sets (line 2).

Every time we handle an arrival event we check if a server is available or not. If one is available, we add a "0" entry to the waiting time record, $W_m \leftarrow [W_m, 0]$, else we add a new monitor $W \leftarrow [W, 0]$, initially being zero because up to *now*, the instant at that the customer arrives, no waiting time has accumulated (lines 7,8). Whenever we jump to the next event the monitors are updated by adding t_i to all monitors, $W \leftarrow W + t_i$ (line 4). If after a departure a customer is taken from the queue by

Waiting time monitor extension

```
 1: — initialise —
 2: ...  W = [ ]; W_m = [ ];  ...                    ▷ initialise monitors and sample arrays
 3: — next event —
 4: ...  W ← W + t_i;  ...                           ▷ increment current waiting times
 5: — arrival —
 6: if (out = Inf) then  ...
 7:       W_m ← [W_m, 0];  ...                       ▷ if immediately served add "0" waiting time to sample
 8: else  W ← [W, 0];  ...                           ▷ else initialise a new waiting-time monitor
 9: end if  ...
10: — departure —
11: if (X > 0) then  ...
12:    W_m ← [W_m, max(W)];                         ▷ add accumulated waiting time to sample (assuming FIFO queue)
13:    W ← [W \ {max(W)}];  ...                     ▷ and remove the finished monitor
14: end if  ...
15: — normalise and return —
16: return  ...  W_m  ...                           ▷ return recorded sample
```

scheduling its departure time, we push the content of the according waiting time monitor to the waiting time sample $W_m \leftarrow [W_m, \max(W)]$ and remove this monitor instance, $W \leftarrow [W \setminus \{\max(W)\}]$ (lines 12,13). Removing the correct waiting time monitor is not straightforward, particularly if other then the FIFO queueing discipline are applied. See section 3.4.3 for details on how to locate the monitor to be recorded and subsequently removed.

When the simulation terminates, the waiting time sample has a size n equal to the number of clients that entered and left the system while the monitoring flag m was set to `true`, and note that this is not equal i or j. The former may contain times from customers served after m became `false`, the latter customers that entered prior m became `true`. We only record the waiting times of events which entirely falls within the monitoring interval. The mean and other statistical properties can be extracted from the recorded sample as already described for the system state record X_m:

$$\mathrm{E}[T_w] = \frac{1}{n}\sum_{i=1}^{n} w_i = \texttt{mean}(W_m) \qquad \Delta_w = \texttt{confid}(W_m, 95)$$

$$\sigma_w = \sqrt{\mathrm{E}[T_w^2] - \mathrm{E}[T_w]^2} = \texttt{std}(W_m) \qquad c_w = \frac{\sigma_w}{\mathrm{E}[T_w]} = \frac{\texttt{std}(W_m)}{\texttt{mean}(W_m)}$$

All queueing system related mean values can again be calculated according to the cyclic dependence depicted in figure 3.75, using the same equations depicted therein, now starting with $\mathrm{E}[T_w]$. Likewise, though only for FIFO queueing and only if the arrival and service processes are independent of the queueing processes, latter jointly referring to the closely related system filling X, flow time T_f, waiting time T_w and queue filling Q, the variances can be calculated accordingly.

Choosing a queueing discipline

To implement the option to select a queueing discipline we add a *policy* string to the parameters handed over to the simulation procedure. We use a string variable for its intuitiveness. A *switch* command structure chooses the customer selection based on this policy string (line 6). Practically we do not need to search the maximum or minimum. The waiting time monitor array W is maintained in a strict order; arrivals are always attached to its end. Consequently, the longest waiting customer is the first in the array (line 8), and the least waiting is the last one (line 11). The random selection does not simplify (line 14). Note that the code snipped presenting this, also shows how we remove the monitor of the scheduled customer from the monitors array (lines 9,12,16).

Implementation of different *queueing disciplines*

```
 1: — initialise —
 2: — next event —
 3: — arrival —
 4: — departure —
 5: if (X > 0) then    ...
 6:     switch(policy)                                  ▷ choose a queueing discipline
 7:         case("FIFO")
 8:             W_m ← [W_m, W(1)];                      ▷ add first monitored to waiting time sample
 9:             W ← W(2 : end);                         ▷ remove first waiting time monitor
10:         case("LIFO")
11:             W_m ← [W_m, W(end)];                    ▷ add last monitored to waiting time sample
12:             W ← W(1 : end−1);                       ▷ remove last waiting time monitor
13:         case("RAND")
14:             index ← ceil(rand(1) length(W));        ▷ choose a waiting monitor index
15:             W_m ← [W_m, W(index)];                  ▷ add chosen one to waiting time sample
16:             W ← W(1 : index−1 , index+1 : end);     ▷ remove chosen waiting time monitor
17:     end switch
18:     out ← feval(.. );    ...
19: end if    ...
20: — normalise and return —
```

By the way, two independent but in certain cases redundant monitors are very useful to check their implementation. For scenarios where the monitored processes are in theory related as depicted in figure 3.75, the resultant statistical characteristics should approximate each other if extracted and calculated via one or the other recorded sample.

3.4.4 Finite system size introducing blocking

Already simulating virtually infinite queues the introduction of an upper bound on the system size is no challenge. All what is essentially required to reject customers in case the queue is full, is to verify if the current system state X is less than the bound s specifying the finite system size (line 6). The system size s is to be provided as another input parameter. If an arriving customer cannot be accepted, meaning added to the queue because it is full, the event passes without a change in the current system state X as if the event has not occurred.

Finite system size and *blocking process* extension

```
 1: — initialise —
 2: ...   B = 0; B_m = [ ];    ...                    ▷ initialise monitor and sample array
 3: — next event —
 4: ...   B ← B + t_i;    ...                         ▷ increment current inter-blocking time
 5: — arrival —
 6: if (X < s) then                                   ▷ if space is available
 7:     ...                                           ▷ handle arrival as usual (accepted → increment X)
 8: else                                              ▷ else reject arrival
 9:     B_m ← [B_m, B];                               ▷ add accumulated inter-blocking time to sample
10:     B ← 0;                                        ▷ reset inter-blocking time monitor
11: end if    ...                                    ▷ generate next arrival event etc.
12: — departure —
13: ...   no change    ...
14: — normalise and return —
15: return    ...   B_m    ...                       ▷ return recorded sample
```

In addition to realising the correct behaviour, we also need to consider the *blocking probability* P_b, which is the most important property of finite systems together with the mean waiting time. Already counting the arrivals and departures (in i and j respectively) this simply would result from the relative difference, $P_b = \frac{i-j}{i}$. However, for a complete simulation study it is expedient to also record the process in order to have access to the higher moments as well. Particularly, if blocked customers become handed over to some other system, the blocking process defines the arrival characteristics to the other system, and thus is often referred to as *overflow process* T_o.

To gather the process we collect the inter-blocking times T_o in the blocking sample B_m, and monitor the inter-blocking times via B, initialised, empty and zero respectively (line 2). Every time blocking occurs, causing an arriving customer to become rejected, we add to the blocking sample the inter-blocking time that has accumulated since the last time a customer became blocked, $B_m \leftarrow [B_m, B]$, (line 9) and reset the inter-blocking time monitor, $B \leftarrow 0$, (line 10). Identical to the departure and waiting time monitors, the blocking monitor is incremented by the inter-event time, $B \leftarrow B + t_i$, whenever the simulation proceeds to the next event (line 4).

The returned sample B_m can be statistically evaluated in the same way as the others before:

$$P_b = \mathrm{E}[T_o] = \frac{1}{n}\sum_{i=1}^{n} b_i = \texttt{mean}(B_m) \qquad \Delta_{Pb} = \texttt{confid}(B_m, 95)$$

$$\sigma_o = \sqrt{\mathrm{E}[T_o^2] - \mathrm{E}[T_o]^2} = \texttt{std}(B_m) \qquad c_o = \frac{\sigma_o}{\mathrm{E}[T_o]} = \frac{\texttt{std}(B_m)}{\texttt{mean}(B_m)}$$

Note, blocking occurs rarely if the load is not approaching the system capacity. Consequently, is the size of the sample B_m commonly quite small and the confidence intervals become huge, particularly at low loads where the blocking probability P_b approaches zero. If a reliable characterisation of the *overflow process* T_o is required, the simulation should not be terminated prior a sufficient sample size for B_m is reached. But be careful, smooth arrival and service distributions, in combination with sufficiently sized queues, can actually result in no blocking at all.

3.4.5 Finite customer population - the *Engset setting*

The up to here presented and commonly assumed simulation generates a next arrival whenever an arrival event occurs. This represents the so called *Erlang setting*, and assures that there always is a next arrival scheduled. For finite customer populations this approach cannot be used. To simulate latter correctly, we need to define a source per potential customer, and schedule arrivals customer by customer. For the Engset setting, where customers cannot generate arrivals while being in the system, the instant to generate the next arrival are the blocking and the departure events.

To implement finite populations we first extend the arrivals variable into an array that offers a dedicated *time till next arrival* for every single source (line 2). When updating all times before handling the current event in the *next event* section, the time passed has to be subtracted from all next arrival times in *in* (line 4). Obviously, the arrival rate has to be specified per customer, and not for the system as common with the Erlang setting. Please refer to section 3.2.4 for details on the resultant issues concerning the system load definition.

In case of an arrival event we need to know which source it refers to (line 6). Note that for the Erlang setting the index is always '1', which equals the index of a scalar value if represented as vector or matrix. With the Engset setting we must not generate a next arrival if the current arrival is accepted (line 9), whereas in case the arrival is blocked, we have to schedule a next arrival in any case (line 12). For sources that are currently in the system (busy), the next arrival is scheduled after their departure (line 18). Here we may use the index of any currently busy source (line 17) because all sources are assumed to behave identically, such that their actual index is irrelevant. Finally, note that for the Erlang setting an $in(1) = Inf$ indicates the end of the simulation loop. With the Engset setting the simulation loop ends with $\texttt{all}(in(:) = Inf)$.

Finite customer population – the *Engset* setting

```
 1: — initialise —
 2: ...  in = [0,0,..,0]  ...                          ▷ initialise customer sources array of size c
 3: — next event —
 4: ...  mathit in_i ← mathit in_i − t  ∀_i  ...       ▷ update by t as usual, now for all sources
 5: — arrival —
 6: ...  c_id ← find(in ≤ eps)  ...                    ▷ get index of current arrival's source
 7: if (X < s) then  ...                               ▷ handle arrival as usual (accepted → increment X)
 8:     if not Engset then in(c_id) ← feval(..)        ▷ Erlang setting: schedule the next arrival
 9:     else  in(c_id) ← Inf                           ▷ Engset setting: schedule no next arrival
10:     end if  ...
11: else  ...                                         ▷ handle blocking as usual (rejected → X unchanged)
12:     in(c_id) ← feval(..)  ...                     ▷ schedule next arrival (here also as usual)
13: end if  ...
14: — departure —
15: ...                                               ▷ handle departure as usual (finished → decrement X)
16: if Engset then
17:     c_id ← find(in = Inf)                         ▷ get index of a currently busy source
18:     in(c_id) ← feval(..)                          ▷ schedule next arrival from source c_id
19: end if  ...
20: — normalise and return —
21: return  ...                                       ▷ return and evaluate recorded samples as usual
```

3.4.6 Multi servers systems *G/G/n/..*

The extension from *G/G/*1*/..* to *G/G/n/..* is a little more complex than the integration of a limited system size. Besides another input parameter, being the number of servers n, we need to change the *server-is-idle* condition to if$(X<n)$ (line 8). Most challenging, we need to extend the *out* variable to hold the *time-till-departure* t_d for each currently busy server. Not necessarily need the servers be individually treated, they are assumed to be identical in their capacity such that the holding time T_h does not depend on the used server. Therefore, and particularly to cover the infinite server case *G/G/*∞, we define a dynamic *out* array that holds the scheduled departure times only.

Multi server extension

```
 1: — initialise —
 2: ...  out = [Inf];  ...                            ▷ initialise departures array with no departure (Inf)
 3: — next event —
 4: ...  t_i ← min(in, min(out));  ...                ▷ get current inter-event time
 5: ...  out(2 : end) ← out(2 : end) − t_i  ...       ▷ decrement times till departure (not Inf)
 6: — arrival —
 7: ...
 8: if (X < n) then                                   ▷ if a server is available
 9:     out ← [out, T_h(i)]  ...                      ▷ add time till departure for the just arrived customer
10: else  ...                                         ▷ else add arrived customer to waiting or block it
11: end if  ...                                       ▷ generate next arrival event etc.
12: — departure —
13: ...  index ← out ≤ eps                            ▷ get the index of the current departure
14:     out ← [out \ out(index)]  ...                 ▷ remove currently occurred departure from out-array
15: if (X > n) then                                   ▷ if a customer is still waiting for service
16:     ...  out ← [out, T_h(i)]  ...                 ▷ add time till departure of next scheduled customer
17: end if  ...
18: — normalise and return —
```

The currently closest departure event is min(out) (line 4). Every time the simulation proceeds to a next event all the times till departure present in *out* have to be decremented by the current inter-event

time t_i, as done before for the scalar *out* variable, now applied on all elements currently contained in the *out*-array except the first, being *Inf* indicating no scheduled departure (line 5). Whenever a new departure is scheduled, meaning a customer enters service, its holding time $T_h = \texttt{feval}(..)$ is added to the array, $out \leftarrow [out, T_h(i)]$ (lines 9, 16). When a departure occurs, the according element in *out* becomes zero ($out_k \leq eps$) and needs to be removed (lines 13, 14).

Note, contrary to the waiting time array introduced earlier, the *out*-array does not necessarily comprise monotonic increasing entries. Holding times can vary heavily, such that a later added time till departure can be smaller than a previously added. Therefore, the index determination (line 13) cannot be omitted. To use a sorted list is an alternative, but may be excessive given the limited list size $\leq n$. Finally, if two departures happen to occur numerically at the same time, we do not need to care for which of them is handled first; any is as present as the other.

Server utilisation

The varying size of the *out*-array tells the *server state*, being the number of currently occupied servers, which represents the system *utilisation*. To statistically evaluate it we need another sample array U_m, and as monitor we use the existing system filling X. Every time the system state X changes we add the inter-event time weighted server utilisation to U_m. For $(X<n)$ we add $t_i X$ to the sample, and for $(X \geq n)$ we add $t_i n$. The implementation is accordingly simple.

Server utilisation

```
1:  — initialise —
2:  ...   Um = [];   ...                          ▷ initialise utilisation sample array
3:  — next event —
4:  ...
5:  if (X < n) then   Um ← [Um, ti X];            ▷ not all servers are busy
6:  else              Um ← [Um, ti n];            ▷ all servers are busy
7:  end if   ...
8:  — arrival —
9:  — departure —
10: — normalise and return —
11: return   ...   Um/simT   ...                  ▷ return recorded sample
```

For *infinite queueing systems* the $\sum U_m/n$ approximates ϱ, the system load. For *finite systems* the utilisation never becomes 100%, it can only approach *one* if the system is sufficiently overloaded, meaning $\varrho > 1$, causing many blocked arrivals. To avoid very small numbers, the normalisation by *simT* is best done after the summation and not before.

Idle period evaluation

An interesting property not mentioned till now is the distribution of the idle period. The analytic approach to queueing systems has shown that it is the idle period that challenges us most. To some extend it is the counterpart to the *over-flow* process, because the *idle process* refers to the state at the other end of the state transition diagram. However, for a multi server system we need to separate two cases: (a) the *true* idle state, meaning X=0, and (b) the union of all states where at least on server is idle, X<n, when no arriving customer needs to wait for being served.

Note that the *true* idle time $T_{X=0}$, meaning that the queueing system in total was idle, cannot persist longer than one inter-event time. Once the system became idle, the next event must be an arrival. The idle period starts with a departure event that empties the system, and ends with the arrival of the next customer, which immediately becomes served causing a busy server. In between no other event can occur and consequently can an idle period never comprise more then a single inter-event time t_i. If in the end we normalise the sample by the accumulated simulation time *simT* we get the *idle probability*, which equals $1-\rho$ for infinite systems only.

Idle period (a: the system is idle)

```
 1: — initialise —
 2: ...   I_m = [];   ...                          ▷ initialise idle time sample array
 3: — next event —
 4: ...
 5: if (X = 0) then   I_m ← [I_m, t_i];            ▷ the system was idle during the current inter-event time
 6: else              I_m ← [I_m, 0];              ▷ at least one servers was busy during the current inter-event time
 7: end if   ...
 8: — arrival —
 9: — departure —
10: — normalise and return —
11: return   ...   I_m   ...                       ▷ return recorded sample
```

In the other case, where we are interested in the distribution of the periods $T_{X<n}$, during which an arriving customer becomes served immediately such that it experiences zero waiting time, the condition may and often will persist across several inter-event times. Therefore, we need a monitor I to accumulate the times during which this idle condition $X<n$ persists. The implementation seems similar; however, we now add a sample only when the condition changes, which can occurs only when $X=n$. Thus, the accumulated idle-period time is recorded, $I_m \leftarrow [I_m, I]$, and reset, $I \leftarrow 0$, only if it is non-zero, $I > 0$ (lines 7, 8, 9). Evidently, the monitor I again needs to be incremented by t_i alike the others whenever the simulation proceeds to the next event, but only if $X < n$ (line 5). In the end the recorded sample needs to be normalised by the accumulated simulation time *simT* (line 14) in order to consider the not recorded times where all servers were busy.

Idle period (b: at least one server is idle)

```
 1: — initialise —
 2: ...   I = 0; I_m = [];                         ▷ initialise monitor and sample array
 3: — next event —
 4: ...
 5: if (X < n) then   I ← I + t_i;                 ▷ increment current idle-period time
 6: end if   ...
 7: if (X = n) & (I > 0) then                      ▷ some idle-time has accumulated and all servers are now busy
 8:     I_m ← [I_m, I];                            ▷ add accumulated idle-period to sample
 9:     I ← 0;                                     ▷ reset idle-period monitor
10: end if   ...
11: — arrival —
12: — departure —
13: — normalise and return —
14: return   ...   I_m/simT   ...                  ▷ return recorded sample
```

The opposite of this I_m is C_m, the *server saturation statistic*, expressing the likelihood distribution for all servers being busy. Usually we monitor this, because in case of negative exponentially distributed arrivals its mean value equals the probability that an arriving customer has to wait for service $p(i \geq n)$, which resembles the *Erlang_C* formula known from equation 3.13. Practically, we achieve this by adjusting the conditions in line 5 and 7 to $X \geq n$ and $X = n-1$ respectively, and use C to monitor $T_{X \geq n}$ and C_m to record the sample thereof.

3.4.7 Processor sharing - *egalitarian* and *discriminatory* PS

We directly proceed to implementing *discriminatory processor sharing* (DPS) as sketched below and presented in the end of section 3.3.3. Egalitarian PS is a special case of DPS, which evidently may be implemented differently, with less complexity. Please see addendum A.I.2 for the actual code used to perform simulation studies, including streamlined egalitarian PS.

First, we need to handle different flows, which are identified by the assigned weights g_i. Therefore, we extend the next arrival variable *in* into a vector that holds the time-till-next-arrival per flow. In an Engset setting *in* becomes a matrix. In case all flows have Markovian inter-arrival times this is not necessary, we could as well assign arrivals to different flows at the arrival instances according to the flow's arrival probabilities $\frac{\lambda_i}{\lambda}$. However, that approach restricts the utility because it excludes different arrival processes per flow. Second, to do the discriminatory processor sharing we need to monitor the number of present clients per flow. This is achieved by extending the state variable X into a vector as well (line 2). Third, we need to initialise the service sharing factors σ_i. Assuming the system to be idle at the simulation start, prior the transient phase, we initially have $\sigma_i = 1\ \forall_i$ (line 3) because the first arrival will initially get the full service rate $\frac{1}{\mu_i}$.

Processor sharing – *discriminatory* PS

```
1:  — initialise —
2:  ...   in = [0;0;..;0];   X = [0;0;..;0];        ▷ initialise per flow – same size as weights vector g
3:        σ = [1;1;..;1];    ...                    ▷ initial sharing factors – same size as weights vector g
4:  — next event —
5:  ...   out_j ← out_j − t   ∀_j   ...             ▷ update as usual, consider all out-times
6:  — arrival —
7:  if (X_i < s_i) then                             ▷ if space is available, for unbound PS is s_i = Inf
8:      ...                                         ▷ handle arrival as usual (accepted → increment X_i)
9:      if PS or (∑X_i < n) then                    ▷ if processor sharing or idle server available
10:         if PS then h ← 1/(σ_i μ_i)              ▷ calculate scaled holding time
11:         else  h ← 1/μ_i                         ▷ regular holding time
12:         end if
13:         out ← [out, feval(1/λ_i, .., h, .., ...)]   ▷ provisionally schedule the departure
14:     end if
15: else  ...                                       ▷ blocking does not occur with PS, can remain as usual
16: end if  ...                                     ▷ proceed as usual
17: — departure —
18: ...                                             ▷ handle departure as usual (→ decrement X_i, remove from out-array)
19: if (∑X_i > n) then ...                          ▷ no waiting customers exist for PS, can remain as usual
20: end if  ...
21: — post event adjustments —                      ▷ new section!
22: if PS and (∑X_i > 0) then                       ▷ if there are processor sharing clients present
23:     σ_i^new ← g_i / (∑_j g_j X_j)   ∀_i         ▷ calculate new sharing factors
24:     out_j ← out_j σ_i / σ_i^new   ∀_j^(*)       ▷ re-adjust all times-till-departure considering current state change
25:     σ ← σ^new                                   ▷ update all σ_i to current scaling factors (∀_i)
26: else if PS then σ ← [1;1;..;1]                  ▷ reset σ to initial when server become idle
27: end if
28: — normalise and return —
29: return  ...                                     ▷ return recorded samples as usual
```

Next the departure time scheduling needs to be adopted. This is not straight forward because with processor sharing the service rate varies and depends on arrivals prior and during the time spent in the system. Thus, independent of weighted or fair sharing, the actual time-till-departure cannot be determined until the departure occurs. Therefore, we split the departure scheduling into a *provisionally set time-till-departure* calculated when handling the arrival instance (line 13) and the *adjustment of all scheduled departures* performed after handling arrivals and departures, meaning after the system state changed (line 24). Because the adjustment is performed on all clients, we need to set a provisional time-till-departure as if the current arrival had not arrived yet. This happens implicitly because we calculate new scaling factors after event occurrences only (line 23). Scaling all current times-till-departure by the relation previous-to-new (line 24) yields the adjustment required.

In line 25 the new sharing factors become the current sharing factors, and in line 26, which might be negligible, we reset the sharing factors to the initial values whenever the server become idle.

(*) Note that in line 24 we need to re-scale the times-till-departure for all scheduled departures depending on the flow i these belong to. How this index $j \leftrightarrow i$ relation is achieved is not included here in the pseudo code. In the real code we use an out-matrix, where the row number indicates the flow and only one value per column can be set. This utilises already the extension for multiple flows required for the examples of the next chapter. Here, any other $j \leftrightarrow i$ relating would work just as well, for example a second row holding the flow index of the departure scheduled in the same column, but henceforth we imply the matrix implementation.

Service time monitoring

Commonly the service time is a given parameter. Also for processor sharing a service process is required as input parameter. However, with PS it specifies the serving in case of no other clients present. The actual time a client spends in service is effectively its flow time T_f. Monitoring this correctly demands us to consider that PS is not order conserving. Therefore, we need to monitor the time a client remains in service client-by-client, meaning in parallel for all currently present clients.

To do so we initialise a service time monitor S and a cell-array S_m to collect the occurred service times per flow (line 2). Both are dynamically extended, the first entry of S accumulates the entire simulation time, with flow index 0 indicating that it refers to no particular flow. For S_m a *cell-array* is used because this allows differently long records per flow.

Time spent in service

```
 1: — initialise —
 2: ...   S = [0;0]; S_m = { };   ...          ▷ initialise serving monitor and record for it
 3: — next event —
 4: ...   S(2,:) ← S(2,:) + t;   ...          ▷ add inter-event time to all service time monitors
 5: — arrival —
 6: ...
 7: if PS or (∑X_i < n) then                  ▷ arrival can immediately be served
 8:     ...   S ← [S, [i_flow;0]];   ...      ▷ add a time-in-service monitor entry
 9: else   ...                                ▷ waiting, blocking (does not occur with PS)
10: end if
11: ...
12: — departure —
13: ...
14: index ← out ≤ eps                         ▷ get the index of the current departure
15: S_m(S(1;index)) ← [S_m(S(1;index)), S(2,index)]   ▷ record monitored time S(2,index) in S_m(i_flow)
16: S ← [S \ S(index)]                        ▷ remove obsolete monitor from S-array
17: ...
18: — post event adjustments —
19: — normalise and return —
20: return   ...   S_m   ...                  ▷ return all the recorded per flow samples arrays
```

As with all monitors we add the times t that pass in between events to the entries of the monitor S, here all entries in the second row because the first row holds flow indices (line 4). With every arrival we initialise a new monitor by adding the (flow-index, flow-time)-pair $(i_{\text{flow}}, 0)$ to the monitor (line 8). If a departure occurs we first get its index j from the *out* array (line 14), which here equals the corresponding index of the S monitor. Having this, we record the monitored time-in-service in the according S_m record (line 15), before we remove the now finished monitor column from S (line 15).

The returned cell-array S_m contains per flow an array holding all recorded times that clients from the same flow spent in service. These arrays are commonly not equally long, particularly not if the load shares $\frac{\lambda_i}{\mu_i}$ are not the same for all flows. However, every $S_m(i_{\text{flow}})$-array can be statistically

evaluated and yields the individual flow time $T_f(i_{\text{flow}})$ characteristics. This is exactly what we need to evaluate the influence of flows on each other, and the dependence thereof on the assigned weights g_i in case of discriminatory processor sharing. For common queueing models, being FIFO, LIFO, RAND queueing systems, which per server serve one client after the other, the monitored time-in-service characteristics need to equal those of the given service processes.

3.4.8 Implemented *G/G/n/s/X* simulation core and its application

First we show in algorithm 3.2 the *m*-style code of the present time based simulation core realised as an *Octave* function. Such functions are stored as text files, for best compatibility one file per function. The function can be called on demand from the command line as well as other *Octave* functions and routines, if the set search path contains the directory it is filed to. We note, the real plain *m*-code

Algorithm 3.2 Present time based *G/G/n/s/FIFO* simulation core

```
function Xm=presentGGnsSim(a,A,b,B,n,s,sr,st)
   in=0; out=[Inf]; ti=0; X=0; Xm=[]; simT=0; i=0; j=0; k=0; m=false; di=[];
   do
      if (i==st) m=true; elseif (length(Xm)==sr) m=false; endif
      ti=min(in,min(out)); in-=ti; if (length(out)>1) out(2:end)-=ti; endif
      if m Xm=[Xm,ti*X]; simT+=ti; endif
      if (in<=eps) i++; di=[]; %arrival
         if (m || i<=st) in=feval('distrGen',A,1/a(1,1),num2cell(a(1,2:end)){:}); else in=Inf; endif
         if (X<s) X++;
            if (length(out)<=n) out=[out,feval('distrGen',B,1/b(1,1),num2cell(b(1,2:end)){:})]; endif
         else k++; %blocking
         endif
      elseif (length(di=find(out<=eps)(1))) j++; %departure
         out=[out(1:di-1),out(di+1:end)]; X--;
         if (X>=n) out=[out,feval('distrGen',B,1/b(1,1),num2cell(b(1,2:end)){:})]; endif
      else error('GGnSim: a not scheduled event occurred.'); break;
      endif
      if (!length(di) && rem(i-st,5000)==0) % show simulation progress and snapshot
         printf('\n%3.1f %% done: i=%d, j=%d, k=%d; \n',(i-st)/(sr/2)*100,i,j,k);
         printf(' %d: \t %5.4f \t %d \t',fci,in,X); printf(' %5.4f',out(:)); printf('\n');
         printf(' Server occupation: '); printf('%2d ',length(out)-1); printf('\n');
      endif
   until (in==Inf && out==Inf)
   Xm=Xm./(simT/length(Xm));
endfunction
```

actually needs less lines than the according pseudo code, which would result from integrating all changes in algorithm 3.1. We use this form whenever the presentation as pseudo code would be excessive because to our experience the *m*-code is well readable if simple variable names are used. However, common *Octace* programming is based on the functional programming paradigm, which somewhat contradicts object oriented programming.

The function presented as algorithm 3.2 actually contains all changes to model common single stage queueing systems with any queue size and server count, including infinite for both as it is required to simulate *G/G/*∞. Having a dedicated representation for infinite (*Inf*) is a very utile feature here. However, for conciseness the code does not contain the finite customer population (Engset setting) or processor sharing extensions. Generally note, a simulation core including all options will never proceed faster than a core that is cut down to the needed features. In case of Matlab and Octave, for-loops and recursive functions shall be maximally avoided in order to minimise the required computation time, particularly within the simulation loop.

However, in the last `if-endif` clause the code includes a brief progress monitoring not mentioned so far. This helps to monitor the simulation progress and to estimate the remaining time till completion. Observing this, a computation time issue with huge recorded samples has been detected. To eliminate this marginal problem we may calculate and record mean values per reporting loop instead of the

detailed sample. Doing so has no impact on the mean of the final sample, but the standard deviation and confidence interval calculation can no more be applied. To still have access to former, we can calculate and return a square means sample in addition to a means sample. The standard deviation and coefficient of variation follow from

$$\sigma = \sqrt{\mathrm{E}[X_m^2] - \mathrm{E}[X_m]^2} \qquad \text{and} \qquad c_X = \sqrt{\frac{\mathrm{E}[X_m^2]}{\mathrm{E}[X_m]^2} - 1}$$

where the two variables required, $\mathrm{E}[X_m]$ and $\mathrm{E}[X_m^2]$, are the mean values of the two returned samples X_m and X_m^2, which themselves containing mean values of the actually monitored values and their squares respectively, such that $\mathrm{E}[X_m] = \mathrm{E}[\{\mathrm{E}[X]\}]$ and $\mathrm{E}[X_m^2] = \mathrm{E}[\{\mathrm{E}[X^2]\}]$. Due to the reduced number of elements contained per returned sample the confidence interval for the mean values exceeds that of the complete sample, such that it seems save to use `confid(.. )` still. However, mathematically this is questionable, and any information on higher moments is ultimately lost.

The programming language independent presentation and its detailed discussion comprise the essence of this section. The integration thereof should not be an issue requiring an explicit example here. However, the *Octace* *m*-code including all monitors and samples lastly used is provided in addendum A.I.2. The extension to cover *multiple arrival flows*, as it is required for the studies presented in the next chapter, is primarily achieved by extending the relevant variables into vectors or matrices, where required. Please see the code example presented in addendum A.I.2. Actually, the majority of the program lines are caused by these and related extensions. However, if an individual treatment of different arrival flows is not required, multiple ingress flows can be merged into one flow prior entering the queueing system: either using weighted convolution of different distributions, sums of identical distributions applying the according calculation rules, or an approximately specify a single aggregate ingress flow using appropriate summing of the individual mean rates and coefficients of variation. If the contributing flows were defined by their first two moments, an approximation considering these two moments is equally accurate as any other aggregation method.

Simulation environment used to calculate curves

The simulation core presented above provides results for a single setting only, meaning for a particular set of input variables. Commonly we are interested in the behaviour and performance for some parameter range. Therefore, we need to calculate curves. To approximately achieve a curve we need to simulate a set of input parameters, where only the value shown of the x-axis changes. The bigger the set is, meaning the more points along the curve are simulated, the more confident we can be upon the system behaviour in between simulated points; still, we may not exclude missed outliers. A smart choice upon detail versus effort is inevitable and should consider the limited confidence already introduced by the finiteness of simulation.

How to effectively use the *Octave* simulation core presented in algorithm 3.2 to get curves is presented by the example shown in algorithm 3.3. This is a living routine, meaning that is has to be adapted manually to perform different studies. Contrary to the previous we use the script approach instead of the function definition. If the ".m"-type-ending is attached to the script name it can be executed by simply typing its name (without ".m") in the *Octave* command line, else an execution command needs to precede the file name.

To include several curves in one figure we can repeat the curve calculation for a second parameter range, which for presentation clarity should be restricted to just enough curves depicting the influence of the second parameter. This is achieved by be adding in the `for`-loop the according *GGnSim(..)* calls and the addition of more curves (*mean*(sample$_j$),*confid*(sample$_j$)-pairs) to the curves matrix plotted in the end. Similarly can more parameters be evaluated and plotted if the simulation core provides them, requiring the addition of the according parameters mean/confid-pairs only. Because different parameters cause different sized samples they cannot be returned as a matrix; instead a

Algorithm 3.3 Calculating and plotting curves using the *G/G/n/s/FIFO* simulation core

```
%plot GGnsSim results (presentGGnsCurves)
   steps=0.95:0.001:1.00; log=1;
   curves=NaN(length(steps),3);
   curves(:,1)=steps; %x-axis values
   printf('satarting the loop \n');
   for i=1:length(steps) rho=steps(i); %load points
      printf('\n load: %3.2f \n',rho);
      sample=presentGGnsSim([rho,0.5],'nExp',[1/10,5],'nExp',10,Inf,2000000,500000);
      curves(i,2)=mean(sample(:));
      curves(i,3)=confid(sample(:),95);
   endfor
   printf('ended the loop now saving and plotting \n');
   save(['GGnsCurves',datestr(date,'yymmdd'),'.dat'],"curves");
   %plot the simulated vaules
   errorPlot(curves(:,:));
   h=ishold; if !h hold; endif
   if log semilogy([]); axis([0.95,1,9.99,5000]); else axis([0,1,0,10]); endif
   if !h hold; endif
```

cell-array of samples ($\{X_m, D_m, B_m, \dots\}$) is returned, accessible as sample$(k)(:)$, where k is the index of the parameter sample according to the return value order defined in the *function* definition line within the simulation core, and $(:)$ selects all values contained therein.

To plot the created curves matrix at once the first column needs to hold the x-axis values followed by the columns holding the according y-axis values per curve, column by column. However, the *errorbar* plot command does not support this; the *errorPlot* routine has been written to provide the functionality known from the regular *plot* command.

In cases where more variable parameters need to be evaluated, adjacent figures that show different angles can be used. The above presented routine does not provide this, it produces curves and figures for several given parameters in parallel. To do adjacent figure sets an according number of *Octave* instances can be run in parallel to better utilize an available multi-core processor. *Octave* per se is strictly single threaded such that a sole simulation run cannot utilize multiple processing cores, leaving a multi-core desktop PC or server fully responsible. It is possible, though not recommended, to run more instances of *Octave* in parallel than there are processing cores. A three-dimensional plane could as well be calculated and drawn, but calculation effort and depicting complexity typically exceed the benefit.

Remarks on system behaviour and performance prediction using simulation techniques

Concluding the introduction of the simulation environment used shortly to evaluate stand alone traffic management concepts and mechanism, we should note that simulating a technical mechanism is much easier than its analytic treatment. Firstly, if we know a mechanisms realisation we can always and without doubt imitate it in software. Secondly, strict bounds and rules can be incorporated without restrictions, whereas for an analytic treatment we commonly need to either drastically simplify or entirely ignore any non-linearities.

That simulation is not always the best approach becomes evident if the events that cause an entry in the sample record are very rare. If this is the case, either the sample becomes too small for a reliable statistical treatment or the time required to collect a sufficient sample becomes ridiculously long. Manifold countermeasures to overcome this problem exist, statistical as well as procedural. To be effective they tend to utilise a priory knowledge, which makes them problem specific and cause sensitive. However, we also recognise problems if the system approaches instability. In this case we get sufficiently sized samples, narrow confidence intervals, and no precision issues arise. Still, the simulation results achieved become quite unreliable. This is shown in figure 3.76 together with the plot of a generated system filling sample at $\varrho = 1$. These results were achieved using the precise

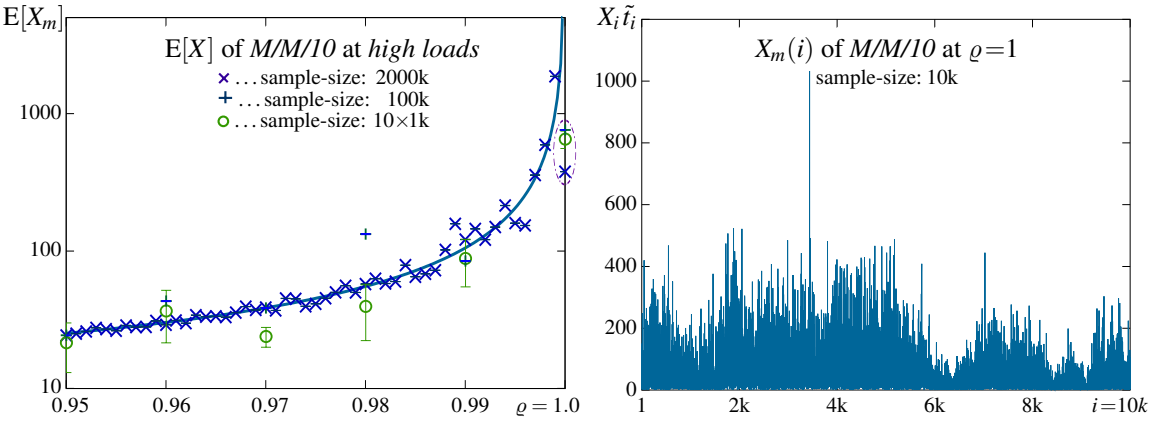


Figure 3.76: Simulating *M/M/10* at loads approaching $\varrho=1$ and an X_m sample record at $\varrho=1$

code presented above. Calculating the 51 samples across the evaluated load region in order to get the $\mathrm{E}[X_m]$ shown lasted approximately 52 hours. Evidently we might split it up and do different sections in parallel to better utilise the multiple cores of the processor. If merging of split results is repeatedly required, a script to merge and draw the saved curves is definitely worth the effort. However, correctly done the result is the same.

Obviously, the simulation core has no problems to simulate $\varrho = 1$ with infinite systems. At this borderline load the simulated system filling X seems to become *chaotic*, meaning that any statistical mean value may result, as for example the encircled results in figure 3.76 show. From some trials performed prior getting the depicted results we got the impression that $\mathrm{E}[X]\,(\varrho{=}1)$ increases with the generated sample size. The shown examples contradict this impression whereas form analysis we perfectly know that it should be infinite.

This discussion may be assumed academic because we know that infinite systems demand $\varrho < 1$. However, this effect also challenges simulation studies at loads close to $\varrho = 1$. For sample sizes which are perfectly sufficient at loads $\varrho < 0.95$, as shown in figure 3.76, the results achieved at higher loads fluctuate notably while the calculated confidence intervals do not correctly indicate that appropriately, they remain closed. For our preferred sample sizes of 100k some of the simulated results above 95% load appear quite off the analytic curve. The situation improves if samples of means and mean squares are used, but only because withthese the confidence intervals open up.

However, it is the transient phase that actually becomes insufficient; for $\varrho = 1$ it would need to be infinite to actually reach the steady state. For the results shown on the left in figure 3.76 a transient phase of 100k arrivals was chosen in any case. This shows that simple simulation studies may not reveal extremes reliably.

A common approach to optimise (minimise) the simulation effort is to repeatedly calculate the confidence interval of the gathered sample and to finish the simulation once it falls below a given threshold, commonly $\Delta_X < 1\%$. However, close to 'chaotic' behaviour this condition may not be met within reasonable time. Consequently, this option should be implemented as short-cut only, never as the sole termination condition. Given the progress reporting loop, its implementation is straightforward. In conjunction with returning mean values only, the condition is to be applied on the confidence interval across the already gathered means and not for the currently gathered values.

Finally, it has to be noted that the routines presented here simulate the model and not a technical mechanism or some realisation. These are provided to analyse models for which a closed form analytic result is not at hand, and to validate the accuracy of approximations proposed in the literature for such unsolved cases. Please proceed to the following chapters for examples on how to model particular traffic management mechanism.

4 Traffic management

Modern communication networks are using different scheduling policies and congestion prevention mechanisms to realise a variety of service classes within the packet switching layer. The intention is to provide reliable *quality of service* (QoS) for traffic flows that demand a certain performance level. Commonly, the applied mechanisms shall not degrade the mean performance of any flow, at least not while $\rho(t) < 1$. Only during intervals with overload, when temporarily $\rho(t) > 1$, they become active and privilege some flows over others. The privileging persists until the *backlog* from the overload has been cleared. Overly differentiation can be disastrous. It is therefore mandatory to study the impact of these mechanisms on any traffic flow, considering the mean impact on the individual traffic flows as well as cross-effects that may relate specific flows. In particular, the mean and variance of the end-to-end flow-times are to be carefully maintained in order to achieve the intended QoS levels without jeopardising the service reliability.

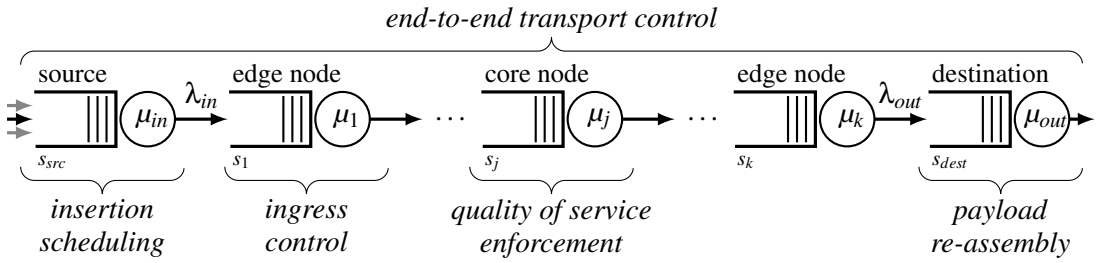


Figure 4.1: Different scopes and locations of traffic management mechanisms

Figure 4.1 depicts the different approaches to traffic management and their location along a flow's path across a network. Transport control and the scheduling of different flows at source nodes are exterior mechanisms, whereas ingress control and quality of service realisation are interior functions of the data transmission network controlled by an operator.

Payload re-assembly at the destination is mentioned because it generates the feedback required for *end-to-end transport control*. The external service rate to process received data packets μ_{out} we commonly model as infinite, as of today the processing speed of end-systems exceeds by magnitudes the capacity of today's access network data channels, defining the line rate μ_k. Assuming so, the payload re-assembly process depends on the inter-arrival time to the destination queue only.

On the other side, μ_{in} states the speed of the up-link from the customer equipment to the edge node. This link may contain a chain of access providing devices, even an entire company LAN if present. Still, common LAN speeds exceed the capacity of today's access network data channels by magnitudes, such that practically the access network's data channel determines the effectively available data transmission rate μ_1.

Egress nodes, being the network's edge node next to the flow receiving destination, typically do not influence the data flow in a way different from that of core nodes. However, if the destination side access network capacity λ_{out} is the least along the path, the egress node will contribute most to the end-to-end delay.

At the ingress side, among the traffic source and the edge node it connects to, we commonly find some means to perform active *access control*. Primarily to avoid collisions but also to distribute the access capacity in a fair way. If the traffic patterns vary heavily or the number of sources is rather unbounded and a jointly accessed shared medium is used, we commonly find demand driven control strategies. The traffic sources communicate their demand to a central control instance, which than assigns the resources to the requesting terminals, such that all are served and no collisions occur (*access scheduling*). This approach is widely used to control wireless as well as wired access networks operated in a collision free shared medium regime. If the traffic pattern is more stable or the number of traffic sources rather small, this two-tier mechanism is better replaced by a one-tier *polling* mechanism, where in principle all terminals are granted access one after the other. Only if a source has nothing left to transmit, it returns or forwards the access granting *token* to the control instance or the next source, respectively. To implement differentiation with polling, an optional scheduling like policy can control (*custom*) how often and for how long the shared resource may be used by the different traffic sources. In the literature we find many strategies to optimise fairness and at the same

Table 4.1: Sharing control strategies

approach		features
demand scheduling	two-tier	global demand awareness → calculated temporary optimal serving schedule ← differentiation by schedule adjustment (*privileging*)
source polling	one-tier	simple, very responsive, inefficient at low loads → implicitly fair resource sharing ← differentiation by *min/max*-bounds (*customs*)

time assure service quality. Basically, perfect fairness and assured service quality for all sources state opposing demands. Therefore, a universal optimum does not exist. The best per-hop sharing strategy is always a compromise that depends on the assumed relevance of the opposing demands. Anyhow, service quality cannot be granted by a single hop, but considerably hampered.

To implement MPLS the *scheduling policies* controlling the local sharing of resources are of primary relevance, at edge and core nodes likewise. The resource sharing among MPLS flows is solely controlled by the local scheduling policies. Every single resource along the entire transmission path needs to contribute its share in a constructive way, such that in the end the intended service quality is achieved – a *chain of individual per-hop contributions* that breaks if one link fails to provide its share. Thus, the local *QoS enforcement* at every involved node is central to achieving an intended end-to-end service differentiation.

Extending the focus from single links to networks, the distribution of traffic across the links and nodes of the network becomes relevant. This is controlled by the *routing algorithm* and the protocols implementing it. Several approaches to achieve efficient *load balancing* are available and in practice used. The topic is huge and rarely studied in relation to queueing models, and thus also not covered. Instead, we assume the aggregate traffic flows transported in between nodes are given (set). A mechanisms to improve the transport performance without changing the present mean load (*smoothing*) is evaluated by means of queueing models in section 4.2.3.

Finally, *ingress control* and *ingress limiting* represent mechanisms applied on the surface of data networks to control the traffic inserted. The former is based on feedback provided by the network and demands a feedback controlled queueing model, whereas latter is commonly applied rather statically, reflecting the paid for *service level agreement* (SLA). In contrast thereto is *admission control* a mechanism to grant or reject access, intended to protect the infrastructure and commonly specified in terms and conditions. Concerning *network feedback* we note that due to the inevitable time lags a detailed but outdated knowledge is inferior to a smartly approximated, problem oriented, location specific, and timely received feedback. Smart data networks better rely on autonomous, locally maintained, globally cooperating *traffic management architectures*.

4.1 Resource sharing

In this section we discuss three concepts widely used with communication networks to smartly handle concurrent resource demands. In contrast to the comparably static multiplexing techniques used on the physical layer, which for example provide space-, time-, frequency-, or code-division-multiplexed channels, the techniques discussed here apply for a shared usage of theses physical channels (in particular their capacity) to either transport potentially more data flows in parallel than channels are available, or to utilize the physical channels in parallel to transport the present flows most efficiently over the joint capacity available. Both scenarios yield a finite multi-server queueing system, where the channels are represented by the n servers and the backload of each flow defines its own queue, as shown in figure 4.2. The dynamic assignment of resources, per load unit, is possible because

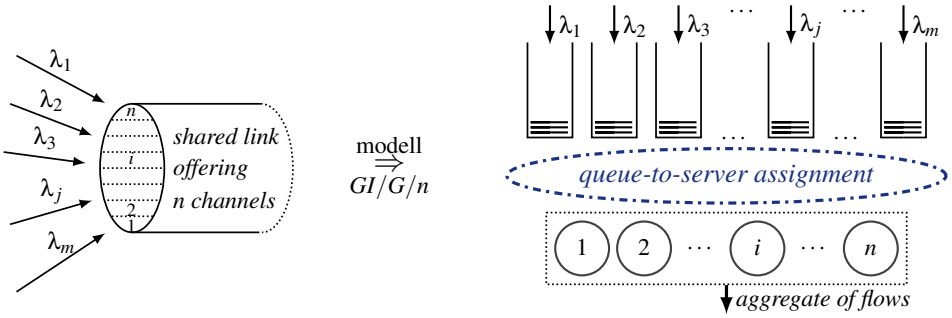


Figure 4.2: Resource sharing: m flows transported over n channels composing a link.

packet switching commonly implies the *store-and-forward* principle. This is technically less efficient than *cut-through* switching, but essential to realise *statistical multiplexing* of dynamic data flows. Today, most traffic flows show a varying momentary load; even voice and video signals became varying due to the content sensitive compression mechanisms now used. Thus, the disadvantage of store-and-forward switching is widely compensated by the efficiency gained from statistical multiplexing.

4.1.1 Egalitarian resource sharing

If the available capacity is equally assigned to populated queues, we achieve egalitarian sharing, which approximates *egalitarian processor sharing* (section 3.3.3). The major difference is that the load is not served in parallel but one-by-one. This has the important drawback that the *head-of-line* blocking caused by huge packets persists. To achieve a better match, which at least partly evades the head-of-line issue, we need a multi-server approach, where the servers, representing the physical transmission channels available, are assigned to queues flexibly. Table 4.2 highlights the two cases to consider, where $j(t)$ states the number of currently populated queues, and $k(t)$ the number of servers

Table 4.2: Assigning n servers to m queues

scenario	serving policy
$j(t) \leq n \;\rightarrow\; k(t) = \frac{n}{j(t)} \geq 1$	serve all populated queues in parallel, in average assigning an equal number of servers $k(t)$ to each
$j(t) > n \;\rightarrow\; k(t) = \frac{n}{j(t)} < 1$	serve n queues in parallel with *one* server each, rotate served queues packet-by-packet (*round-robin*)

assigned to each of the currently non-idle queues.

The number of servers $k(t)$ assigned depends on the number of populated queues only, $k(t) = \frac{n}{j(t)}$, and needs adjustment whenever a formerly idle queue becomes populated or a currently served queue becomes idle. Hardly this can be achieved instantly. But more seriously, the calculated $k(t)$ is

rarely an integer number. However, with some engineering this can be approximated by assigning $\tilde{k}(t) = (\lceil k(t) \rceil, \lfloor k(t) \rfloor)$ to queues and toggling these numbers among queues packet-by-packet, such that in average the calculated $k(t)$ service share is assigned to each populated queue. This costs some performance and is in general not perfect because the re-assignment of more or less servers can only be performed in between packet transmissions. However, it at least approximates the fairness required to achieve egalitarian resource sharing. On the other hand, any $k(t) > 1$ causes another engineering problem, namely the re-assembly of packets at the destination side, particularly challenging if packets are transported bit- or byte-wise spread across parallel channels while the channels used change packet-by-packet. This necessitates buffers and the transmission of some control information along with the packets, which causes even more overhead that again costs some performance.

Transmission overheads and timing issues are commonly not considered on the network layer of packet switched networks. Commonly, they are seen as lower layer costs not related to the problems handled within the network layer. Thus, we no further investigate egalitarian resource sharing and assume henceforth that any real numbered resource share can be assigned to any queue. The performance loss caused by lower layer issues is presumed a priory subtracted in the capacity provided to the network layer. That it may depend on the current load is for simplicity ignored.

In figure 4.3 simulation results for *M/M/9/es* are compared with the performance of perfect egalitarian sharing (processor sharing – PS), which for the evaluated metrics equals *M/M/1* (see section 3.3.3). Evidently, the queue filling depends on the load shares contributed, and therefore

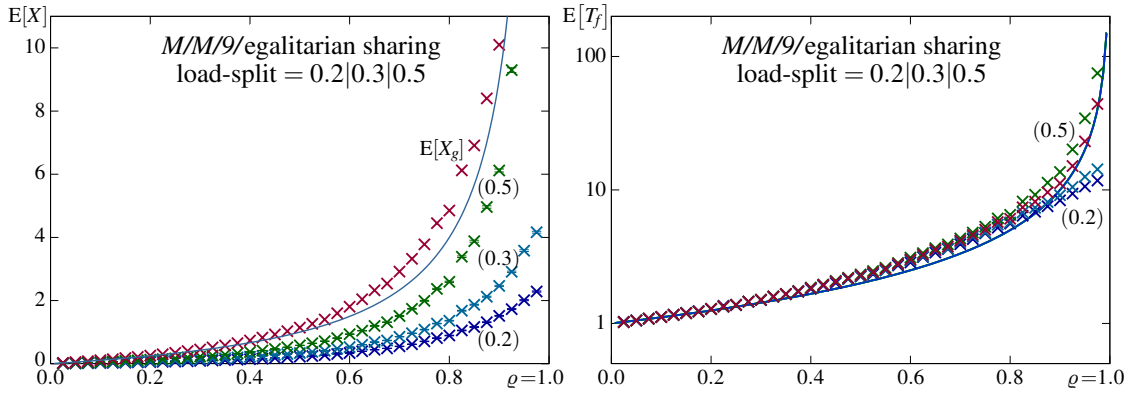


Figure 4.3: Packet based *egalitarian sharing M/M/9/es* simulation results ($\times$) compared to same capacity *M/M/1* system (solid lines), assuming Poisson distributed packet arrivals, negative exponentially distributed holding times (packet lengths), and a load split $\frac{\rho_i}{\varrho} = \{0.2|0.3|0.5\}$

are the mean queue fillings $\mathrm{E}[X_i]$ not identical. The mean system filling $\mathrm{E}[X_g]$ is slightly above the reference curve, which we get for *M/M/1* and *M/M/PS*. Consequently, also the mean flow time $\mathrm{E}[T_f]$ is slightly above the reference curve. However, in contrast to the queue filling, up to quite high load (>0.75) we observe nearly no differentiation of the individual mean flow times $\mathrm{E}[T_f(i)]$. This equals the behaviour of processor sharing *M/M/PS* and a single infinite queue *M/M/1*. Above, for $\varrho > 0.75$, the observed mean flow times $\mathrm{E}[T_f(i)]$ diverge from the ideal case, showing the influence of the head-of-line blocking, and that less filled queues are privileged at high system loads. This behaviour causes a penalty on heavy flows, which effectively occurs when the system is in a critical load state only. Thus, in case the ingress load is controlled by a round-trip-time dependent mechanism, for example TCP, the flows responsible for the high load ($\rho_i > \frac{\varrho}{|i|}$) are affected earlier than flows that cause less than the average load per flow. Commonly, such a behaviour is welcomed because it assures that no flows starve.

For a given number of flows the egalitarian serving strategy can be modelled by a continuous time Markov chain. For presentation clarity we restrict the model depicted in figure 4.4 to two servers and two flows. In theory this can be extended to any number of servers and flows. However,

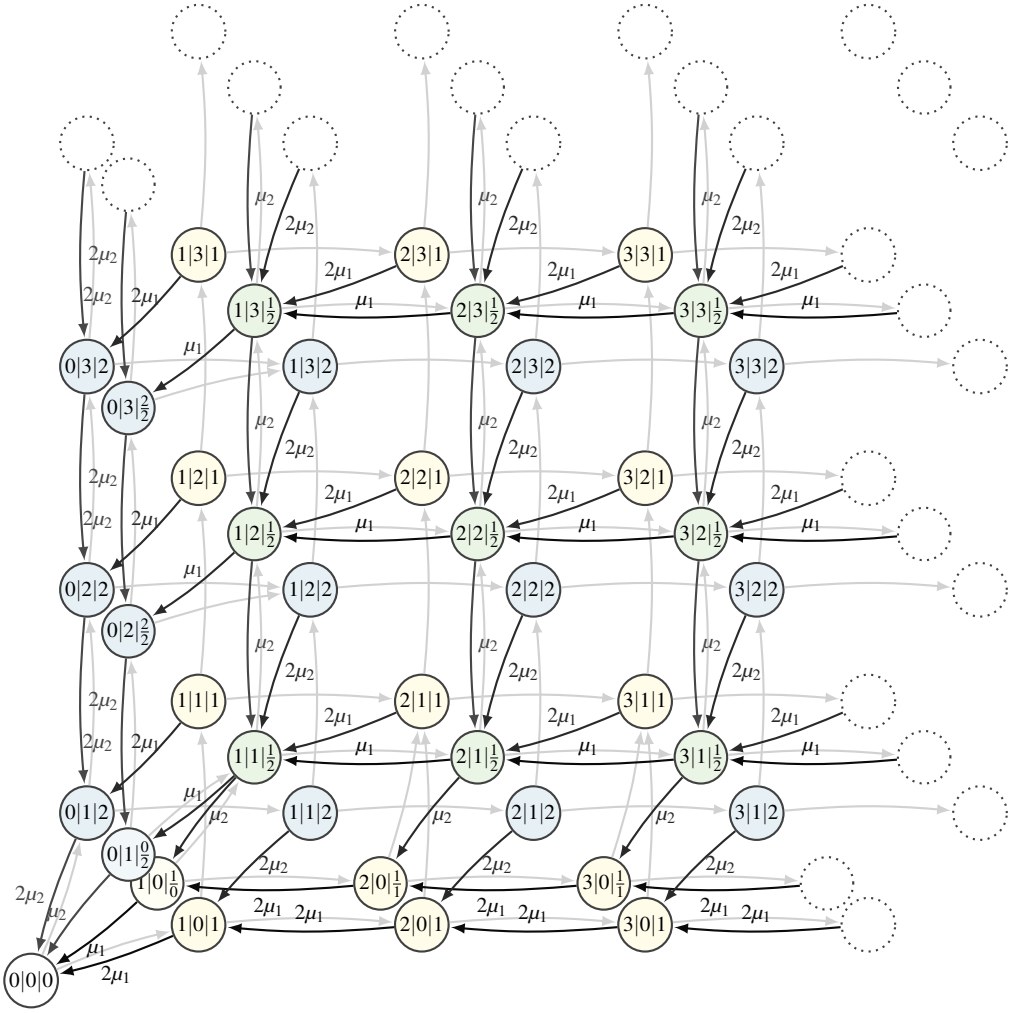


Figure 4.4: *M/M/2/es* continuous time Markov chain model for *two* flows and *infinite* queues (arrival transitions are shown in light grey, negative bent for λ_1 and positive bent for λ_2, as for departures)

for increasing numbers of either, the continuous time Markov chain quickly becomes hard to grasp because all server occupation variants need to be modelled as individual subsets, each itself with dimension equal the number of flows. The number of such subsets required is defined by the number of possible variations with repetitions, which is given by

$$\binom{n_f + n_s - 1}{n_s} = \frac{(n_f + n_s - 1)!}{n_s!\,(n_f - 1)!}$$

where n_f is the number of flows, and n_s the number of servers n. For twelve flows and eight servers we get 75 582 twelve dimensional subsets. In addition, also the edges are rather tricky. At the edges we need to separate states where the same server occupation pattern occurs but packets are served by different number of servers. For example, one packet of a flow may be served by one servers, another by three, depending on the number of servers assigned to the packet that last departed. These different states cannot be merged into a state where two packets are served by four servers because the paths to the idle system are different – as shown in figure 4.4. Thus, for practical system assumptions it may be intractable to state the continuous time Markov chain. The mechanism per se is evidently not burdened by this state explosion; it comprises $n_f(t)$ queues and one selection mechanism managing all the servers. Thus, the real system's complexity grows linear with the number of flows n_f, and is independent of the number of servers.

Returning to the two-flows case we realise that it is not possible to define repeating levels such that we could use the *matrix geometric method* to solve this infinite system. If we group states such that the total number of customers rises by one per level we get an *increasing number of states, level by level*, and thus, there exist no repeating levels. Grouping into levels where only the number of clients of one flow increases, results in repeating still infeasible levels because of the infinite number of states per level.

To approximately evaluate such a system analytically, we may try to *decompose the state diagram* into tractable subsystems. We identify four regions: the idle system (0), two edge regions where only one queue is occupied and served (1), and the remaining region where all queues are served in parallel (2). Assuming equal holding times $\mu_i = \mu$, we can approximate the round-robin service in region (2) by an *M/M/1* system with a virtual service rate $\mu^{(n_f)} = \frac{n\mu}{n_f}$, according to the egalitarian service policy specified in table 4.2. The edge regions (1) can be approximated by an *M/M/1* system with service rates $\mu^{(1)} = n\mu$, where we marginalise the special states with different server assignment per packet served. Note that for now the super-positioned indices (j) stated with service rates indicate the *number of flows served* rather than the index i of the flow served or the j-th moment. In the idle state (0) the service rate $\mu^{(0)}$ is undefined and irrelevant.

Based on this decomposition the *M/M/2/es* system performance might be approximate by weighted sums of the performance metrics we get from the simple *M/M/1* systems identified, at according loads. The weighting factors $w_i^{(j)}$ are the normalised probabilities for the system being in a region. Assuming independent and identically distributed arrivals and $\mu = 1$, we get

$$
\begin{aligned}
p_{[0,0]} &= p(X_1 = 0 \cap X_2 = 0) = (1 - \tfrac{\rho_1}{1-\rho_2})(1 - \tfrac{\rho_2}{1-\rho_1}) \\
p_{[1,0]} &= p(X_1 > 0 \cap X_2 = 0) = \tfrac{\rho_1}{1-\rho_2}(1 - \tfrac{\rho_2}{1-\rho_1}) \quad \rightarrow \quad w_1^{(1)} = \tfrac{p_{[1,0]}}{p_{[1,0]}+p_{[1,1]}} \\
p_{[0,1]} &= p(X_1 = 0 \cap X_2 > 0) = (1 - \tfrac{\rho_1}{1-\rho_2})\tfrac{\rho_2}{1-\rho_1} \quad \rightarrow \quad w_2^{(1)} = \tfrac{p_{[0,1]}}{p_{[0,1]}+p_{[1,1]}} \\
p_{[1,1]} &= p(X_1 > 0 \cap X_2 > 0) = \tfrac{\rho_1}{1-\rho_2}\tfrac{\rho_2}{1-\rho_1} \quad \rightarrow \quad w_1^{(2)} = \tfrac{p_{[1,1]}}{p_{[1,0]}+p_{[1,1]}}, \; w_2^{(2)} = \tfrac{p_{[1,1]}}{p_{[0,1]}+p_{[1,1]}}
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \quad \mathrm{E}[X_i]_{M/M/2/es} &\approx w_i^{(1)}\,\mathrm{E}[X_i]^{(1)}_{M/M/1} + \tfrac{\rho_i}{\rho_1+\rho_2} w_i^{(2)}\,\mathrm{E}[X_i]^{(2)}_{M/M/1} = \rho_i\left(\tfrac{w_i^{(1)}}{1-\rho_i} + \tfrac{w_i^{(2)}}{1-\rho_1-\rho_2}\right) \\
\mathrm{E}[T_{f_i}]_{M/M/2/es} &\approx w_i^{(1)}\,\mathrm{E}[T_{f_i}]^{(1)}_{M/M/1} + w_i^{(2)}\,\mathrm{E}[T_{f_i}]^{(2)}_{M/M/1} = \tfrac{w_i^{(1)}}{1-\rho_i} + \tfrac{w_i^{(2)}}{1-\rho_1-\rho_2} = \frac{\hat{x}_i}{\rho_i}
\end{aligned}
$$

where the last relation complies with Little's law $N = \lambda T$. However, figure 4.5 reveals that this is no

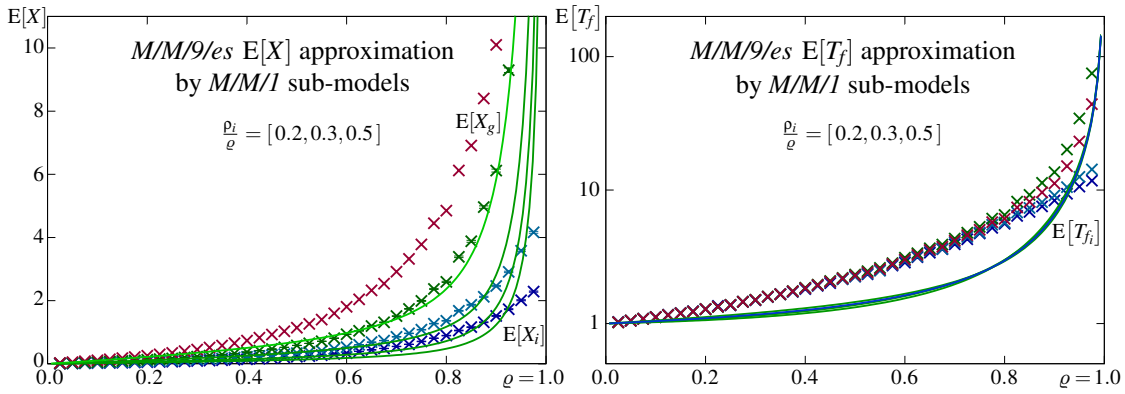


Figure 4.5: *M/M/n/es* system approximation by independent *M/M/1* models

viable approach to an effectual approximation. The result heavily overestimates the true performance, indicating that the interaction among flows is not sufficiently covered by the *M/M/1* sub-models that result per queue-occupation-pattern $\vec{x}$.

Alternatively, we can assume a service process that integrates different service rates $\mu^{(j)}$ with according probabilities, being the normalised weights $w_i^{(j)}$, with $\sum_{j=1}^{n_f} w_i^{(j)}=1\ \forall_i$, as shown in figure 4.6. This model resembles a hyper-exponential service process, which is likely, given the sharing approach,

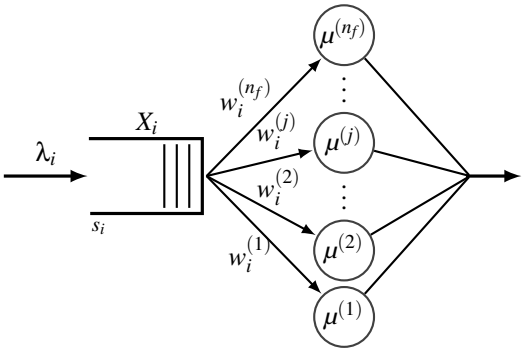


Figure 4.6: Approximate per flow model resulting from decomposing an *M/M/n/es* system

but also reveals a critical shortcoming of the model: the model assigns service rates randomly to adjacent packets, which the true system never does. Thus, higher moments derived via this model will most likely not approximate those of the true system.

To get the required parameters we extend the decomposition to more flows and servers by adding regions for every possible queue-busy-or-idle vector $\vec{x} = [\langle 1,0\rangle | X_k>0]$. Their probabilities $p_{\vec{x}}$ yield the weights $w_i^{(j)}$, and the number of busy queues j gives their virtual service rates $\mu^{(j)}$

$$p_{\vec{x}} = \prod_{X_k>0} \frac{\rho_k}{1-\sum_{i\neq k}\rho_i} \prod_{X_k=0}\Big(1-\frac{\rho_k}{1-\sum_{i\neq k}\rho_i}\Big) \quad \rightarrow \quad w_i^{(j)} = \frac{\sum_{|\vec{x}|=j} p_{\vec{x}}}{\sum_{X_i>0} p_{\vec{x}}} \qquad \text{and} \qquad \mu^{(j)} = \frac{n\mu}{j} \tag{4.1}$$

where to achieve the weights per flow i, the sum in the nominator covers all regions where the number of busy queues equals j, $|\vec{x}| = j$, and where the flow's queue is not idle, $X_i > 0$. The denominator provides the normalisation to $\sum_{j=1}^{n_f} w_i^{(j)}=1$. The idle state $p_{\vec{0}}$ is a unique region, where $w_i^{(0)}=0\,\forall_i$ as required. Also unique is the fully busy region $p_{\vec{1}}$, where $X_k>0\,\forall_k$. However, for latter we get finite weights $w_i^{(n_f)}$, which are not equal due to the flow dependent normalisation by $p(X_i>0) = \sum_{X_i>0} p_{\vec{x}}$. If we assume equal μ_i, as in equation 4.1, the service rates $\mu^{(j)}$ do not depend on the present flows, only on their number j. With unequal service rates μ_i this does not hold, and for each possible region (queue-busy pattern $\vec{x}$) an individual service rate $\mu_{\vec{x}}^{(j)}$ and weight $w_{\vec{x}|X_i>0}^{(j)}$ needs to be calculated, and individually included in the model as dedicated service phase.

To calculate the approximate performance metrics we either apply the *M/G/1* results (*Pollaczek-Kintchin mean value formulas*) presented in section 3.1.3, together with the known formula for the first two raw moments of the hyper-exponential service time, $\mathrm{E}[T_h]_{H_k} = \sum \frac{\alpha_j}{\mu_j}$ and $\mathrm{E}[T_h^2]_{H_k} = 2\sum \frac{\alpha_j}{\mu_j^2}$, each summed over all service-time options, as outlined in section 2.1.2, or use the *Matrix geometric method* presented in section 3.1.6. For finite queues the *Matrix analytic method* presented in section 3.2.3 is applicable. However, note that the approximate analytic models are to be solved for the load $\rho_{H_k(i)} = \lambda_i\, \mathrm{E}[T_h]_{H_k(i)}$, where both, the mean holding time $\mathrm{E}[T_h]_{H_k(i)}$ and the arrival rate λ_i, are model specific: $\mathrm{E}[T_h]_{H_k(i)} \geq \frac{1}{n\mu}$ according to the hyper-exponential service process, and $\lambda_i = \frac{1-p_0(i)}{\mathrm{E}[T_h]_{H_k(i)}}$ with $p_0(i) = p(X_i{=}0)$ being the probability that the queue of flow i is idle, such that in the end we have $\rho_{H_k(i)} = 1 - p_0(i) = p(X_i{>}0)$. The different results for different flows, $\mathrm{E}[X_i]$ and $\mathrm{E}[T_{f_i}]$, result from the different models analysed per flow given the different weights $w_i^{(j)}$ found per flow i (equation 4.1).

How well this approximate model predicts the true performance metrics, in particular the mean system filling $\mathrm{E}[X_i]$ and the mean flow time $\mathrm{E}[T_{f_i}]$, is shown in figure 4.7. The approximate model's

mean metrics (solid lines) are achieved using the Pollaczek-Kintchin mean value formulas together with the known moments of the hyper-exponential service process sketched above. Doing so, we get

$$\mathrm{E}[X_i] \approx \lambda_i \,\mathrm{E}[T_h]_{H_k(i)} + \frac{\lambda_i^2 \,\mathrm{E}[T_h^2]_{H_k(i)}}{2(1-\lambda_i \,\mathrm{E}[T_h]_{H_k(i)})} = \lambda_i \left(\sum \frac{w_i^{(j)}}{\mu^{(j)}} + \frac{\sum \frac{w_i^{(j)}}{\mu^{(j)^2}}}{\frac{1}{\lambda_i} - \sum \frac{w_i^{(j)}}{\mu^{(j)}}} \right) \tag{4.2}$$

$$\mathrm{E}[T_{f_i}] \approx \mathrm{E}[T_h]_{H_k(i)} + \frac{\lambda_i \,\mathrm{E}[T_h^2]_{H_k(i)}}{2(1-\lambda_i \,\mathrm{E}[T_h]_{H_k(i)})} = \sum \frac{w_i^{(j)}}{\mu^{(j)}} + \frac{\sum \frac{w_i^{(j)}}{\mu^{(j)^2}}}{\frac{1}{\lambda_i} - \sum \frac{w_i^{(j)}}{\mu^{(j)}}} = \frac{\hat{x}_i}{\lambda_i} \tag{4.3}$$

where all summations are taken over all flow counts, $j = 1 \,..\, n_f$.

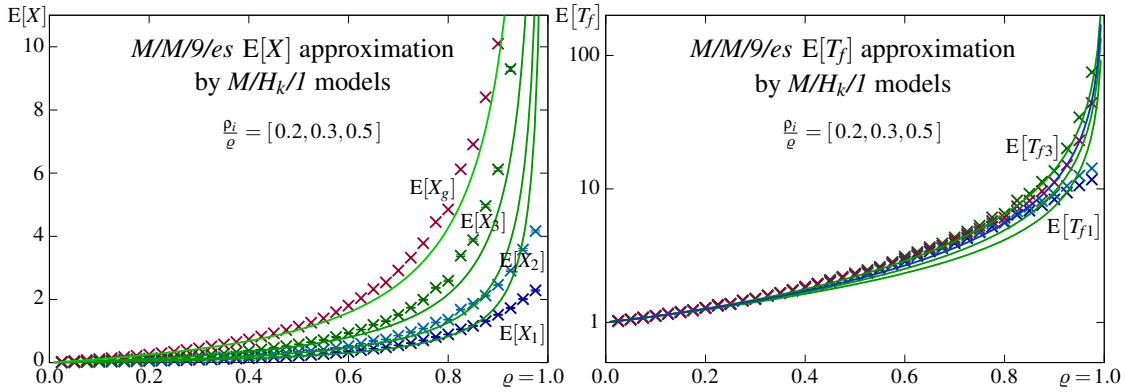


Figure 4.7: *M/M/n/es* system approximation by flow specific $M/H_k/1$ models

Comparing the approximate analytic results (solid curves) with the simulations results ($\times$), we recognise that the model fits quite well at low loads, whereas at medium loads it underestimates the true metrics, and finally at very high loads it overestimates the true metrics. Thus, it neither provides an upper nor a lower bound. We conclude, that here decomposition is a viable but in its utility problematic approach to analysis simplification.

Actually, the required calculation of all $p_{\vec{x}}$ implies a permutation, which rises a scalability issue. However, the complete continuous time Markov chain representing an *M/M/9/es* system serving three flows comprises 55 three dimensional subsets, which for a finite system with queue size $s_i = 10$ results in around $55 \times 3^{10} = 3\,247\,695$ states. Numerically this can be solved; however, the effort to set-up, cross-check, and finally solve such huge equation systems may not be appreciated if sufficiently accurate results can be achieved more efficiently. The decomposition outlined yields 2^{n_f} queue filling patterns, each representing a potential serving performance, defining a phase of the H_k model. In case of flow independent holding times this number reduce to n_f phases only. The number of servers n has no impact on the complexity of the decomposed models because the n servers are replaced by a single virtual server with load dependently varying holding time.

Hereinafter we generally assume that a link's capacity is available in total, meaning the aggregate capacity of the link is not split into individual physical channels. This simplification allows us to restrain the system modelling to the more convenient *GI/G/*1 queueing models, and in addition, maybe even more importantly, it renders the models and evaluations based thereon *independent of the physical layer* implementation. Thus, the models developed henceforth do not dependent on how the data transmission among nodes is actually realised. Figure 3.5 in section 3.1.1 and figure 3.21 in section 3.1.5, comparing single server and multi server systems with identical total capacity, give an idea on the expectable deviation from reality.

4.1.2 Traffic prioritisation

The most common and straightforward discipline to implement differentiation is prioritising the packets of one flow over those of other flows. Typically, more than two levels are required, and while the highest priority does achieve guaranteed performance, the performance of intermediate levels is difficult to predict if the load of higher privileged loads is not static.

Concerning traffic prioritisation we note three basic issues, which are summarised in table 4.3. First, the *global equilibrium law* states that the average performance cannot be improved. This implies

Table 4.3: Implicit issues of prioritisation

issue	consequence
global equilibrium law	improved average performance is not possible
state dependent $\mu_i(t)$	load dependent correlation among departing flows
flow discrimination	*neutrality* and *best effort* concepts are violated

that nodes always offer their full capacity to serve all demands as good as possible. Consequently, if the current load is entirely comprised of loads belonging to the same priority class, the achieved performance must equal the average system performance, irrespective of the chosen priority level. If differently prioritised loads are processed, the improvement experienced by higher prioritised loads causes less prioritised loads to experience worse than average performance.

Second, an *improvement comes at some price*. First, the overhead required to implement prioritisation will slightly deteriorate the average system performance. The other price to be paid is the introduced correlation among differently prioritised flows. An initially smooth low priority flow departs as a bursty flow because its serving is stochastically interrupted by the highly prioritised load batches that randomly build up in the high priority queue, being the respective queue filling $x_i(t)$. These batches cause temporary starvation, which in case of overload becomes persistent.

Finally, prioritisation contradicts the *neutrality concept*, which enforces the egalitarian *golden rule* to serve all demands as good as possible, also known as the *best effort* scheme. This may be considered optional. However, simple network planning and quality assessment strategies may be based on fairness and equality assumptions. In consequence, the achieved performance my diverge considerably from the expected performance, if a discrimination mechanism is not considered in sufficient detail by the applied planning and quality assessment tools.

Concerning lower layer packet transmission, we should note that the integrity of a transmitted packet is essential for its processing at the receiving node, in particular for error detection and correction mechanisms, being prime features of digital communication. Thus, simple pre-emption demands service restart, causing increased network load and less efficiency. A scheme to resume service, meaning to receive packets in timely displaced data chunks, might be implemented. However, specific signalling and a sufficient guard time seem inevitable to separate the data belonging to the pre-empted packet from that of the pre-empting packet. The introduced time lag likely corrupts the marginal benefit of pre-empting a comparably short service interval. Pre-emption, as commonly applied, seems primarily an option for long holding times, where signalling is effective more or less instantaneously.

Strict priority systems with infinite queueing space

Practical queue sizes can be very large, causing an intractable number of states, particularly if also many priority levels (classes) exist. In this case we can approximate the performance by infinite queues, assuming that losses due to queue saturation are negligible. This approximation is feasible for a total load $\rho<1$ only, where $\rho = \sum_{i=1}^{m} \rho_i$, $\rho_i = \frac{\lambda_i}{\mu}$, and m is the number of priority classes, assuming independent and identically distributed service times T_h. For $\rho<1$ the probability of n

waiting customers decreases exponentially with increasing n, such that for very large queues the blocking probability for $\rho \ll 1$ is negligibly small.

We thus can approximate a *strict priority queueing* (SPQ) system by an $M/G/1$ queueing system. The distribution of arrivals among different service classes i, $i = 1,2,..,m$, is given by $p_i = \frac{\lambda_i}{\lambda}$, where $\lambda = \sum_{i=1}^{m} \lambda_i$, and we assume here increasing indices for ascending priority levels. To derive the performance per service class i we use the remaining work approach and note that for the chosen *queue leaving discipline* the number of clients in the system equals the current queue filling $X(t)=Q(t)$. The average waiting time $t_W = \mathrm{E}[T_W]$ is independent of the scheduling discipline and comprises two factors: the mean remaining service time t_r of the currently served client plus the service times $\mathrm{E}[T_h] = \frac{1}{\mu}$ of the waiting clients $n(i)$, which per service class contributed $\frac{n(i)}{\mu}$ to the waiting time. We may thus write

$$t_W = t_r + \sum_{i=1}^{m} \frac{n(i)}{\mu} = t_r + \sum_{i=1}^{m} \frac{\lambda_i t_W(i)}{\mu} = t_r + \sum_{i=1}^{m} \rho_i t_W(i) \tag{4.4}$$

where $t_W(i)$ is the mean waiting time of class i clients until serving starts, and $n(i) = \lambda_i t_W(i)$ applies Little's formula to get the number of waiting clients excluding the currently served. For $M/G/1$ we derived in section 3.1.3 $\mathrm{E}[T_W] = \frac{\lambda}{2(1-\rho)} \mathrm{E}[T_h^2]$ (equation 3.34) and using this with equation 4.4 we get

$$t_W - t_r = \sum_{i=1}^{m} \rho_i t_W(i) = \rho t_W = \frac{\rho \lambda \mathrm{E}[T_h^2]}{2(1-\rho)} \tag{4.5}$$

which is independent of the scheduling discipline and mathematically expresses that whenever by prioritisation the mean waiting time $t_W(i)$ of some classes is in average reduced, this must be compensated by increased mean waiting times $t_W(j)$ for other less privileged classes.

To calculate the mean *residual server occupation time* t_r, being the time until the next client can be served, we use the remaining service time $\mathrm{E}[T_r] = \frac{\mathrm{E}[T_h^2]}{2\mathrm{E}[T_h]} = \frac{\mu}{2} \mathrm{E}[T_h^2]$, which applies when a client is actually served, and multiply it with the probability that a client is served $p_{i>0} = 1 - p_0 = \rho$:

$$t_r = \rho \frac{\mu}{2} \mathrm{E}\left[T_h^2\right] = \frac{\lambda}{2} \mathrm{E}\left[T_h^2\right] \tag{4.6}$$

To calculate the *mean waiting times* $t_W(i) = \mathrm{E}[T_W(i)]$ for different service classes we need to consider that lower prioritised classes are never served prior the one we investigate, while higher prioritised clients are served prior the observed until all their queues are idle. Latter can be split in two groups: those that in average are already waiting when the test client arrives, $n_j = \lambda_j t_W(j)$, and those that arrive while the test client is waiting, $m_{ji} = \lambda_j t_W(i)$. The mean waiting time for class i can thus be composed as

$$\begin{aligned} t_W(i) &= t_r + \sum_{j=i}^{m} \frac{n_j}{\mu} + \sum_{j=i+1}^{m} \frac{m_{ji}}{\mu} = \\ &= t_r + \sum_{j=i}^{m} \frac{\lambda_j t_W(j)}{\mu} + \sum_{j=i+1}^{m} \frac{\lambda_j t_W(i)}{\mu} = t_r + \sum_{j=i}^{m} \rho_j t_W(j) + t_W(i) \sum_{j=i+1}^{m} \rho_j \end{aligned} \tag{4.7}$$

where $j \geq i$ occurs because we assume ascending class indices for increasing priority and FIFO for clients of the same class. To cover other ordering conventions, the summation bounds need to be adjusted accordingly. Finally we rewrite equation 4.7 sucht that the mean waiting time $t_W(i)$ can be calculated recursively, starting with the highest level $t_W(m)$:

$$\begin{aligned} & t_W(i) - t_W(i+1) = \rho_i t_W(i) + t_W(i) \sum_{j=i+1}^{m} \rho_j + t_W(i+1) \sum_{j=i+2}^{m} \rho_j \\ & t_W(i) \left(1 - \sum_{j=i}^{m} \rho_j\right) = t_W(i+1) \left(1 - \sum_{j=i+2}^{m} \rho_j\right) \qquad \Rightarrow \qquad t_W(i) = t_W(i+1) \frac{1 - \sum_{j=i+2}^{m} \rho_j}{1 - \sum_{j=i}^{m} \rho_j} \end{aligned} \tag{4.8}$$

With $i=m$ we get from 4.7 $\quad t_W(m) = t_r + \rho_m t_W(m) \quad \Rightarrow \quad t_W(m) = \dfrac{t_r}{1-\rho_m}$
and applying 4.8 can recursively calculate all t_W:

$$t_W(m-1) = t_W(m)\frac{1-\sum_{j=i+2}^{m}\rho_j}{1-\sum_{j=i}^{m}\rho_j} = \frac{t_r}{1-\rho_m}\cdot\frac{1}{1-\rho_m-\rho_{m-1}} = \frac{t_r}{(1-\rho_m)(1-\rho_m-\rho_{m-1})}$$

$$t_W(m-2) = t_W(m-1)\frac{1-\rho_m}{1-\rho_m-\rho_{m-1}-\rho_{m-2}} = \frac{t_r}{(1-\rho_m-\rho_{m-1})(1-\rho_m-\rho_{m-1}-\rho_{m-2})}$$

$$t_W(m-3) = t_W(m-2)\frac{1-\rho_m-\rho_{m-1}}{1-\rho_m-\rho_{m-1}-\rho_{m-2}-\rho_{m-3}} = \frac{t_r}{\left(1-\sum_{j=m-2}^{m}\rho_j\right)\left(1-\sum_{j=m-3}^{m}\rho_j\right)}$$

Generalised, the *mean waiting time per class i*, $t_W(i)$, results:

$$t_W(i) = \frac{t_r}{\left(1-\sum_{j=i+1}^{m}\rho_j\right)_+\left(1-\sum_{j=i}^{m}\rho_j\right)_+} = \frac{\frac{\lambda}{2}\,\mathrm{E}[T_h^2]}{\left(1-\sum_{j=i+1}^{m}\rho_j\right)_+\left(1-\sum_{j=i}^{m}\rho_j\right)_+} \tag{4.9}$$

With some care equation 4.9 may be applied even for $\rho>1$ when we replace a negative multiplier in the denominator by zero, indicated by $(..)_+$, such that positive infinite waiting times result for all classes for which $\sum_{j=i}^{m}\rho_j>1$ occurs. These classes are in average never served, which is known as *starvation*, and constitutes a major drawback of SPQ systems.

Finite SPQ systems

The strict priority discipline prefers higher prioritised flows over less prioritised traffic flows, unaware of the current load level. In case all queues are finite, the system remains stable even for overload, and theoretically, complete starvation never occurs. However, in case $\sum_{j=i}^{m}\rho_j>1$ most packets from flows with priority $\leq i$ will be lost due to saturated queues, causing an effective service starvation. A serious data transmission service cannot be maintained in such a situation.

To model finite SPQ we assume that the packets of different flows enter individual priority queues according to some per flow control information provided in-line (header field) or a priori (signalling plane). Thus, different priorities are not assumed to be assembled into a single flow. Per queue the applied discipline is first-in first-out, such that all packets within a queue are peers. This is briefly sketched in figure 4.8 and the processing of packets follows a rather simple rules set:

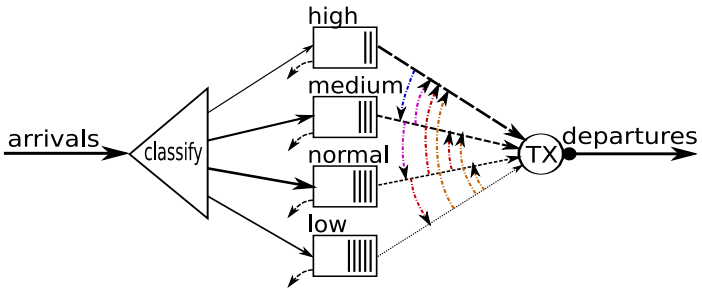


Figure 4.8: Strict priority queueing – SPQ

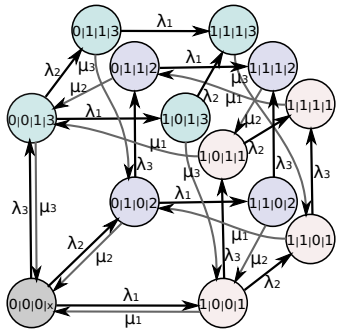
- serve the highest populated priority queue until it becomes empty,
- serve next lower queue until empty or higher priority queue become populated,
- finish packet transmission before switching to higher priority queues.

The processing of a packet is commonly not interrupted because with communication networks it does not make sense to interrupt the transmission of a packet once it has been started. Thus, the

server is not preempted, and conseuently becomes the serving of the higher priority packet delayed by the recurrence-time required to finish the transmission of the packet currently served.

To achieve a utile SPQ model we need to assume at least three priority levels. Two levels would be sufficient to study the performance of the highest and the lowest priority, because from the viewpoint of either, the other may be approximated as the aggregation of many levels. However, the most interesting levels are the intermediate ones. In addition, due to assuming non-preemptive service, the lower priority traffic cannot be neglected as in the preemptive case assumed widely and the literature [96, 97].

Covering three priority levels (queues) results in a three-dimensional state transition diagram. In addition, we need to consider which queue is currently served, splitting the states representing equal queue filling into three potential sub-states. In consequence becomes the state transition diagram quite complex and the resultant Q-matrix grows cubic. Figure 4.9 shows the smallest possible state transition diagram, where the queues provide space for a single client per priority level only, waiting



$$Q = \begin{bmatrix}
-a_1 & \lambda_1 & \lambda_2 & \lambda_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mu_1 & -a_2 & 0 & 0 & \lambda_2 & 0 & \lambda_3 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mu_2 & 0 & -a_3 & 0 & 0 & \lambda_1 & 0 & 0 & \lambda_3 & 0 & 0 & 0 & 0 \\
\mu_3 & 0 & 0 & -a_4 & 0 & 0 & 0 & \lambda_1 & 0 & \lambda_2 & 0 & 0 & 0 \\
0 & 0 & \mu_1 & 0 & -a_5 & 0 & 0 & 0 & 0 & 0 & \lambda_3 & 0 & 0 \\
0 & \mu_2 & 0 & 0 & 0 & -a_6 & 0 & 0 & 0 & 0 & 0 & \lambda_3 & 0 \\
0 & 0 & 0 & \mu_1 & 0 & 0 & -a_7 & 0 & 0 & 0 & \lambda_2 & 0 & 0 \\
0 & \mu_3 & 0 & 0 & 0 & 0 & 0 & -a_8 & 0 & 0 & 0 & 0 & \lambda_2 \\
0 & 0 & 0 & \mu_2 & 0 & 0 & 0 & 0 & -a_9 & 0 & 0 & \lambda_1 & 0 \\
0 & 0 & \mu_3 & 0 & 0 & 0 & 0 & 0 & 0 & -a_{10} & 0 & 0 & \lambda_1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mu_1 & 0 & -a_{11} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mu_2 & 0 & 0 & 0 & 0 & -a_{12} & 0 \\
0 & 0 & 0 & 0 & \mu_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -a_{13}
\end{bmatrix}$$

Figure 4.9: Minimal SPQ: state transition diagram and Q-matrix for one packet per queue

or in service, due to the leave-after-service regime assumed. The Q-matrix of this minimal system has size 13×13, which is huge compared to the 2×2 matrix of a single queue with s=1 (*M/M/1/1*). As before with *egalitarian sharing*, there exist no repeating levels of equal size and internal transitions, as it would be required to apply the matrix analytic method (section 3.2.3).

To evade the scaling issue we decompose the system into sub-models that approximately cover the behaviour. We recognise that the server is not available to a queue whenever a higher priority queue is populated. The server *goes on vacation* until all higher priority queues are again emptied. Thus, the approximate model of choice is the vacation model.

A core problem is the vacation period. It is composed of a varying number of service phases, its duration is *mixed Erlang* distributed, if we assume priority independent holding times $\mu_i = \mu$, or generalised mixed Erlang else. However, in [97] it is shown that with high accuracy this vacation period can be approximated by a Cox-2 model.

Here, we approximate it with a single negative exponentially distributed phase, for simplicity and to outline the principle. An extension to Cox-distributed vacation times is straightforward, integrating the parameter fitting presented in [97] instead of the rather simple probabilistic approach we use with the vacation model depicted in figure 4.10. Anyhow, this sub-model does show repeating levels, and thus, it can be solved using the *Matrix Analytic Method* introduced in section 3.2.3.

The arrival process in our example comprises a single Markov phase, which can be replaced by any *phase type* distributed process.

$$r_{A_i} = \begin{pmatrix} \lambda_i \end{pmatrix} \quad \alpha_A = \begin{pmatrix} 1 \end{pmatrix} \quad q_A = \begin{bmatrix} 0 \end{bmatrix}$$

$$D_0(A_i) = \begin{bmatrix} -\lambda_i \end{bmatrix} \qquad D_1(A_i) = \begin{bmatrix} \lambda_i \end{bmatrix}$$

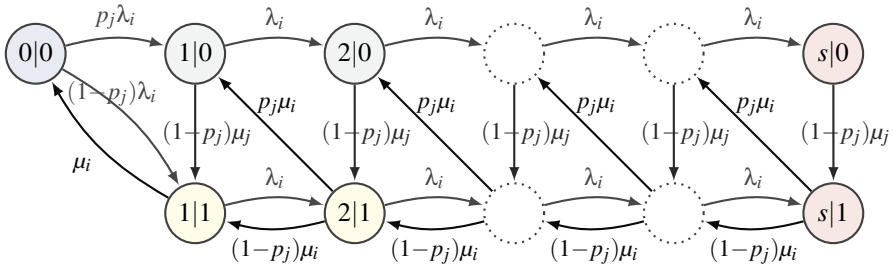


Figure 4.10: A first vacation based approximate sub-model, state indices $(X_i|i_{\text{served}})$

The service process is defined by the used vacation model,

$$r_{S_i} = \begin{pmatrix} \mu_j \\ \mu_i \end{pmatrix} \quad \alpha_{S_i} = \begin{pmatrix} p_j \\ 1-p_j \end{pmatrix} \quad q_{S_i} = \begin{bmatrix} 0 & 1-p_j \\ 0 & 0 \end{bmatrix}$$

$$D_0(S_i) = \begin{bmatrix} -(1-p_j)\mu_j & (1-p_j)\mu_j \\ 0 & -\mu_i \end{bmatrix} \qquad D_1(S_i) = \begin{bmatrix} 0 & 0 \\ p_j\mu_i & (1-p_j)\mu_i \end{bmatrix}$$

where $\mu_j = \sum_{j<i} \frac{x_j}{\sum_{j<i} x_j} \mu_j$ is the presence weighted mean service rate of present higher priorities, and $p_j = \sum_{j<i} P(x_j > 0)$ is the probability that higher priority queues are populated (at any time).

Based on this general *per-priority* model we can set up individual sub-models per priority queue and recursively calculate the probability p_j, for each sub-model representing queue i, and required to solve the next lower priority queue's model. Starting with the highest priority queue, for which $p_j = 0$, and repeating this until the least priority's sub-model is solved, yields the results shown in figure 4.11. Comparing the analytic results (curves) with the simulation results ($\times$) we recognise

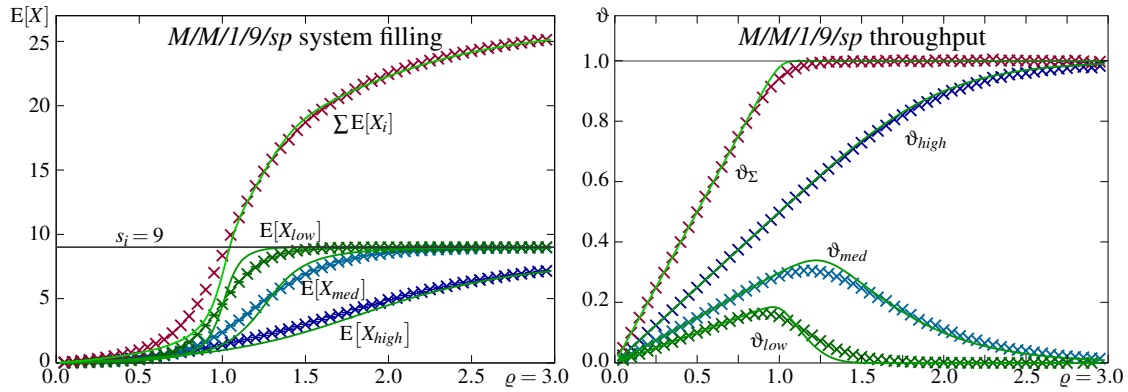


Figure 4.11: First *M/M/1/9/sp* approximation results, calculated (solid lines), simulation results ($\times$), equal holding times ($\mu_i = \mu$), load split $\frac{\varrho_i}{\varrho} = \{0.5|0.3|0.2\}$ for decreasing priorities

that the approximate models tend to over-estimates the performance. For lower priorities the over-estimation is followed by an under-estimation, causing the approximation to yield neither an upper nor a lower bound.

The divergence is likely caused by missing the non-preemption related time-lag of one recurrence time in case a lower priority queue is served when a customer of priority i arrives to an empty queue i. To cover this we have to add a *delayed service* row $(i|*)$, as shown in figure 4.12. In addition to the previous model we now also need the conditional probability that the other queues are idle $p_0^* = \prod_{j \neq i} p_0(j)$, the net arrival rate of load entering higher priority queues $\lambda_j^* = \sum_{j>i} (1-P_b(j))\lambda_j$, and the presence weighted mean service rate of lower priorities $\mu_j^* = \sum_{j>i} \frac{x_j}{\sum x_{j>i}} \mu_j$. Also note, this per-queue model no more resembles a pure *M/Ph/1/s* system. Still, we have the level structure and

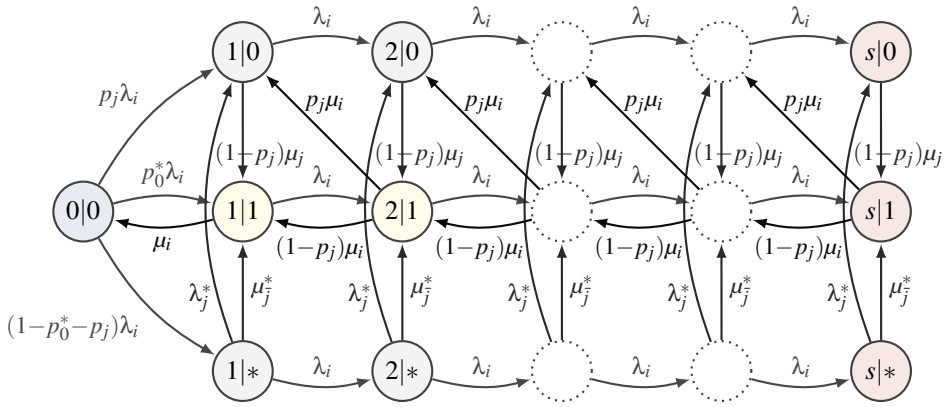


Figure 4.12: Approximate sub-model considering non-preemption, state indices $(X_i|i_{\text{served}})$

can define the sub-matrices $A_{0,1,2}$ and B_s using adjusted $D_{0,1}(S_i)$ matrices. The boundary transitions differ, and the level sub-states include now a transient state $(X_i|*)$ that is not reached from repeating states, only from the idle state.

$$D_0(S_i) = \begin{bmatrix} -(1-p_j)\mu_j & (1-p_j)\mu_j & 0 \\ 0 & -\mu_i & 0 \\ \lambda_j^* & \mu_{\bar{j}}^* & -\lambda_j^* - \mu_{\bar{j}}^* \end{bmatrix} \qquad D_1(S_i) = \begin{bmatrix} 0 & 0 & 0 \\ p_j\mu_i & (1-p_j)\mu_i & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

For negative exponentially distributed arrivals with mean λ_i we get

$$A_1(i) = \begin{bmatrix} -\lambda_i - (1-p_j)\mu_j & (1-p_j)\mu_j & 0 \\ 0 & -\lambda_i - \mu_i & 0 \\ \lambda_j^* & \mu_{\bar{j}}^* & -\lambda_i - \lambda_j^* - \mu_{\bar{j}}^* \end{bmatrix} \qquad A_2(i) = \begin{bmatrix} \lambda_i & 0 & 0 \\ 0 & \lambda_i & 0 \\ 0 & 0 & \lambda_i \end{bmatrix}$$

$$A_0(i) = \begin{bmatrix} 0 & 0 & 0 \\ p_j\mu_i & (1-p_j)\mu_i & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B_s(i) = \begin{bmatrix} -(1-p_j)\mu_j & (1-p_j)\mu_j & 0 \\ 0 & -\mu_i & 0 \\ \lambda_j^* & \mu_{\bar{j}}^* & -\lambda_j^* - \mu_{\bar{j}}^* \end{bmatrix}$$

and the lower boundary sub-matrices $B_{0,1,2}$, all in accordance to figure 4.12.

$$B_0(i) = \begin{bmatrix} 0 \\ \mu_i \\ 0 \end{bmatrix} \qquad B_1(i) = \begin{bmatrix} -\lambda_i \end{bmatrix} \qquad B_2(i) = \begin{bmatrix} p_j\lambda_i & p_0^*\lambda_i & (1-p_0^*-p_j)\lambda_i \end{bmatrix}$$

As before, using the derived sub-matrices we can set up individual models per priority level and solve these. However, the newly introduced probability p_0^* and rate $\mu_{\bar{j}}^*$ result from all the other models, including lower priority's sub-models. Therefore, we need to assume initial values for $p_j, p_0^* \in [0\,..\,1]$, $\mu_{\bar{j}}^* \sim \bar{\mu}$, and solve the *system-of-models* iteratively. We tried different initial values, including the extremes, and they all lead to the same results. In the end we use $p_j = 0$, $p_0^* = 1$ as starting point for the lowest load evaluated, and the values reached at the previous load point as initial values for the next load point. We stop iterating when for all flows the system fillings x_i, the blocking rates δ_i and the throughputs ϑ_i stabilised sufficiently, meaning $\Delta_{x_i,\delta_i,\vartheta_i} < 10^{-12}\ \forall_i$.

The results of the iterative calculation match the simulations results nicely up to a total system load of $\sim 75\%$ ($\varrho \leq 0.75$), as shown in figure 4.13. Up to this load level the convergence constantly worsens, meaning that the number of iterations per load point increases from 10^- to 50^+. This is observed nearly independent of initial values chosen and the requested precision. However, above this load, convergence is no more achieved if we set the precision to $\Delta < 10^{-15}$. Obviously, for

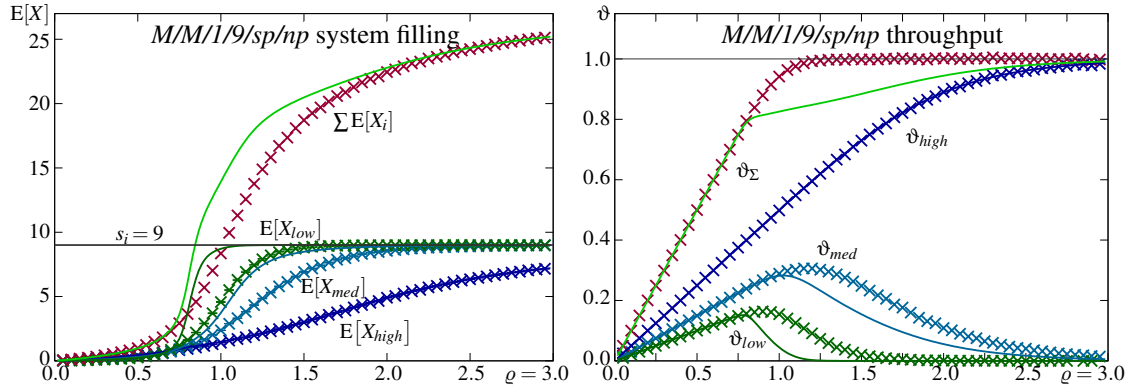


Figure 4.13: Extended *M/M/1/9/sp/np* approximation, calculated (solid lines), simulation results (×), equal holding times ($\mu_i = \mu$), load split $\frac{\rho_i}{\varrho} = \{0.5|0.3|0.2\}$ for decreasing priorities

these loads the iteratively calculated state probabilities of the sub-models fluctuate in behind the 12th decimal place. Above $\varrho = 1$ the convergence problem vanishes and convergence is very quickly reached, as shown in figure 4.14. Although numerical calculation precision issues were not entirely

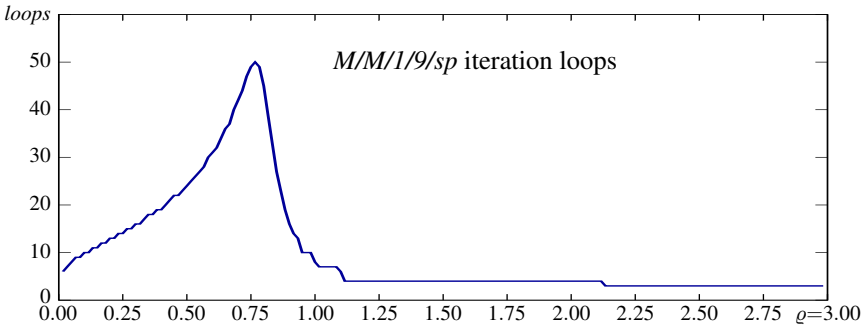


Figure 4.14: Iteration loops required for second *M/M/1/9/sp* approximation example (figure 4.13)

excluded, it is unlikely that these are responsible here. More likely is in this load region the system of sub-models not perfectly stable, meaning the joint solution space is not entirely concave. The iteration than may get stuck, toggling in between two close local minima. The fluctuation in the 12th decimal place alone cannot deteriorate the results as considerably as shown in figure 4.13. The sharp bend on the throughput trace also indicates that here some systemic/methodic issue snaps in.

Comparing the divergence among analytic results (curves) and simulation results (×) with that of the previous approximation we recognise that the approximation quality is in average worse than that of the previous, less complex approach. However, taking a closer look on the divergence caused by this approach, we recognise that it yields a *conservative performance prediction*, particularly visible for the flow times shonw in figure 4.15. The approximate results calculated with this model never promise better performance than what the true (simulated) system actually offers. In that sense, this model appears to be applicable as a performance predictor for QoS provisioning.

Figure 4.15 shows the flow time of SPQ for two different load-splits. For the same system load very different mean flow times T_f result for the different priority levels, and the simple model assuming pre-emption underestimates in particular the mean flow time of the highest priority, $\mathrm{E}[T_{f,high}]$. As intended by SPQ, the medium priority service experiences considerably less delay at the same system load, when less higher prioritised load is present. If we compare $T_{f,med}$ for different load splits but at the same high priority load, for example at $\varrho = 1$ on the left and at $\varrho = 2.5$ on the right, than the much heavier system load causes a slightly increased mean flow time.

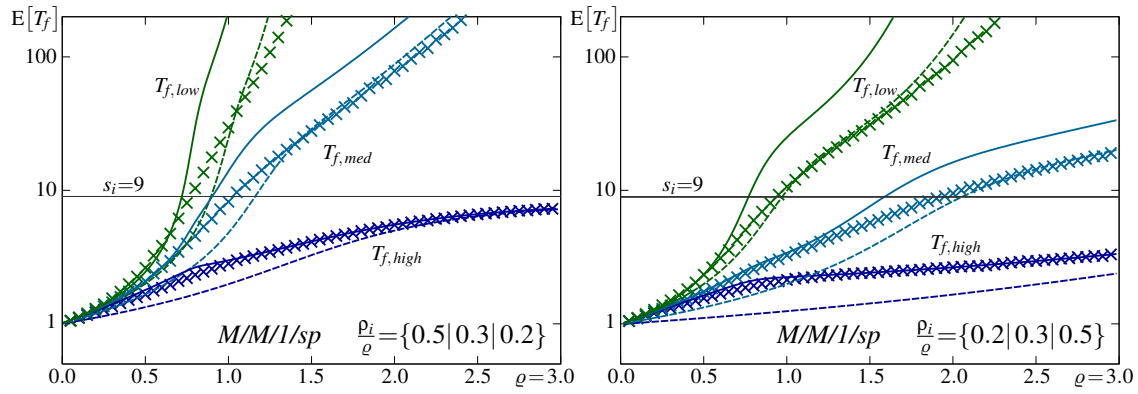


Figure 4.15: *M/M/1/sp* mean flow time $\mathrm{E}[T_f]$, simple (dashed lines) and conservative (solid lines) approximation, simulation results ($\times$), equal holding times ($\mu_i = \mu$), different load splits.

Many more evaluations of this resource sharing scheme are possible, we could for example analyse the departure characteristics. The amount of freely selectable parameters makes a qualified selection of system parameter yielding expressive results relevant for most parameter settings difficult. The experienced problems with approximating the true behaviour show that the behaviour of such internally entangled systems is no breeze. Here we end the analysis of strict priority systems and draw some conclusions. Please refer to the rich literature on the topic, which spans decades of research, for example [96–101] on more sophisticated methods to approximately analyse *M/G/n/SP* queueing systems.

Noteworthy, prioritisation causes flow-times that approach infinity, even though the system may be finite. This starvation potential, analytically expressed by equation 4.9 and clearly visible from the vanishing throughput shown in figures 4.11 and 4.13, causes that for a communication centric application, where *a minimal performance is required for service survival*, SPQ is rather poor and should be used only when *the total load from prioritised flows never exceeds the available capacity*. For a decentralised implementation this means that *excessive prioritised arrivals need to be discarded*, for example by implementing a policy that guarantees some minimum capacity share for lower prioritised traffic, the opposite of the policy presented in [101]. It can be anticipated that in conjunction with smart transport control, for example TCP, such a policy is beneficial for both, the privileged and the non privileged traffic flows, because it assures decreasing upper flow-time bounds for increasing priority levels. Economically, strict prioritisation is a purblind approach to QoS provisioning if the business models foresee a steady growth of high priority load shares. Only if the relation among priority levels is well maintained, SPQ performs as intended.

The systems discussed next are based on weights rather than hierarchic priority levels and can be adjusted more flexibly to changing situations, centrally via a *network operation centre* (NOC) or distributed and flow centric, granting local privileges to demanding services. They manage the resource shares accessible per flow as foreseen in the so called *software defined networks* (SDN) operation paradigm. Accordingly, the mechanisms discussed in the following widely replaced most other resource sharing mechanisms and represent the current state-of-the-art technique deployed.

4.1.3 Weighted scheduling

A traffic classification based resource sharing system is always bound to an a priori defined number of classes and a rather static configuration of the per-hop-behaviour per traffic class. A valid approach to overcome some shortcomings of this approach is class swapping, meaning to enable different class assignments hop-by-hop. However, to achieve a perfectly flexible resource sharing scheme it is necessary to switch from static traffic classes to traffic stream related weighting factors, extending the

principle of class based queueing to a per flow level. A *higher weight shall grant better service*, and *all flows shall be handled in parallel*, such that they influence each other no more than inevitable.

We already evaluated a model that achieves this. In section 3.3.3 we present *discriminatory processor sharing* (DPS) and mentioned its sibling, *generalised processor sharing* (GPS). However, right from the start we had to commit that *processor sharing models presume infinitely dividable fluid like traffic*. We thus surmised that the models represent optima, which practical resource sharing systems may intend to approach. In this section we introduce and analyse packet scheduling systems that intend to approximate processor sharing on a packet-by-packet basis.

Weighted fair queueing (WFQ)

The basic scheme has been presented as early as 1989 by Alan Demers, Srinivasant Keshav, and Scott Shenke in [102]. A very similar approach based on *virtual clocks* is presented in [103] by Lixia Zhang. Since than several more variants evolved.

Weighted fair queueing (WFQ) offers upper-bound waiting times and has been proven to grant upper bound end-to-end delay, if combined with *leaky bucket* based ingress flow limitation, as noted in [104] and elsewhere. WFQ achieves a weighted distribution of the available serving capacity considering the variable packet size when these arrive, and thereby it approximates generalized processor sharing (GPS) very well. While GPS requires infinitely dividable loads, packets commonly need to be transmitted one-by-one to maintain their integrity, at least logically, on a per-hop perspective. WFQ eliminates the thereby risen head-of-line blocking issue quite effectively by a scheduling policy that smartly considers different packet sizes.

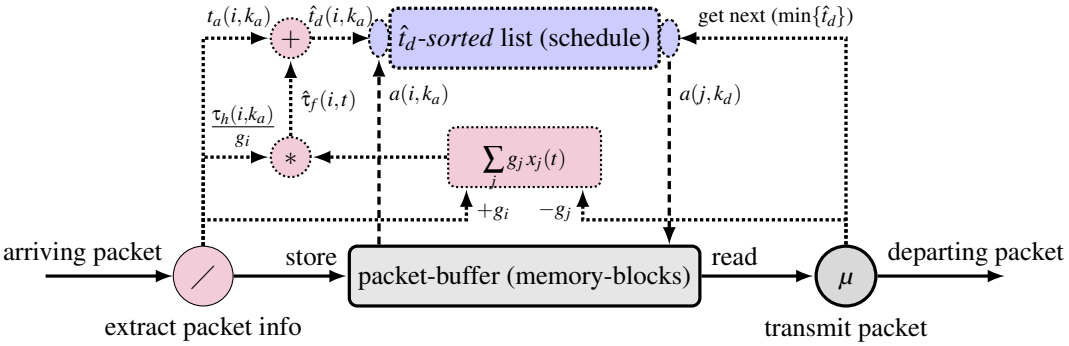


Figure 4.16: Block diagram for weighted fair queueing (WFQ)

To approximately achieve this fluid like sharing performance the scheme depicted in figure 4.16 is commonly applied: When packet k_a with weight g_i arrives, an expected flow time $\hat{\tau}_f(i,t)$ is calculated, based on the current service share $\sigma_i(t)$ (equation 3.108) and the arriving packet's length $\ell_{k_a} = \mu\tau_h(k_a)$

$$\hat{\tau}_f(i,t) = \frac{\tau_h(k_a)}{\sigma_i(t)\,\mu} = \tau_h(k_a)\frac{\sum_j g_j x_j(t)}{g_i\,\mu}$$

where g_i and $\tau_h(k_a)$ are the assigned weight and the holding time of the current arrival k_a from flow i, and g_j and $x_j(t)$ are the weights and the number of scheduled packets of all currently present flows.

The required weighted sum $\sum_j g_j x_j(t)$ results simply from successive adding and subtracting of g_j as arrivals and departures occur, respectively. It does not depend on the current arrival, and can thus be maintained independently in control memory, such that it is not re-calculated arrival-by-arrival. Adding the calculated flow time expectation $\hat{\tau}_f(i,t)$ to the packet's arrival time $t_a(i,k_a)$ we get the *expected departure time*:

$$\hat{t}_d(i,k_a) = t_a(i,k_a) + \hat{\tau}_f(i,t)$$

This $\hat{t}_d(i,k_a)$ is than used to schedule the packet's processing by inserting the address of the memory block buffering the arrived packet in a t_d-sorted *transmission list*. Whenever ready to handle the next packet (k_d), the transmitter of the according out-port picks the first address from this list and transmits the thereby addressed packet, freeing the used memory blocks once transmission is accomplished.

Note, a small packet arriving shortly after a huge one will be scheduled prior the huge one. This anomaly mitigates the head-of-line blocking issue at the price of out-of-order transmission. In general, transmitting packets of different flows not in the order they arrived rises no problem. Only for packets of the same flow this may be an issue. Per flow in-order transmission is granted for equally sized as well as sufficiently spaced packets. Thus, flows with bursty inter-arrival and holding time distribution will be severely affected. However, means to efficiently handle out-of-order transmission should be available at any destination's network terminal in order to support simple services that presume ordered packet reception.

To analyse the performance of WFQ we may either refer to the *generalised processor sharing* (GPS) model [105], or the *discriminatory processor sharing* (DPS) presented in section 3.3.3. Both are fluid policies, where the flow specific resource assignment is achieved by introducing weighting factors g_i that cause an uneven assignment of the total processing capacity μ to the individual present flows i. The service share $\sigma_i(t)=\frac{\mu_i(t)}{\mu}$ is according to [105] lower bounded by

$$\sigma_i(t) = \frac{g_i x_i(t)}{\sum_j g_j x_j(t)} \geq \frac{g_i}{\sum_j g_j} \tag{4.10}$$

where g_j is the weight of flow j, and $x_j(t)$ indicates a flow's presence (1 if present, 0 if not). With DPS $x_j(t)$ states the number of customers of class j currently present (in service). Note that this is the same as if we would open multiple queues for flows with the same weight g_j. However, the bound does not hold for DPS, where j relates to the number of weights and not the number of flows, and as 'bound' it is quite useless because it drops to zero if the number of flows rises to infinity. The true value of equation 4.10 is that it mathematically proves that the relation among granted shares equals that of the weights. Thus, a flow with a smaller weight never gets more shares than one with higher weight, while the full capacity is always provided (*best effort*).

This guaranteed minimum service share allows us to state an upper bound for the mean flow time $\mathrm{E}[T_{f,i}]$ per flow. To do so let us assume a virtual, decomposed single queue *M/G/1* system with service rate $\mu'_i = \sigma_i \mu_i$ based on the minimally granted service share $\sigma_i = \min_t \sigma_i(t) = \frac{g_i}{\sum_j g_j}$, as for example shown in [106]. According to equation 4.10 this service rate is guaranteed to be less than the service share with which the queue is actually served. Applying the results known for *M/G/1* we can thus immediately state an upper bound

$$\mathrm{E}[T_{f,i}]_{M/G/WFQ} \quad \leq \quad \mathrm{E}[T'_{h,i}] + \frac{\lambda_i \mathrm{E}[T'^2_{h,i}]}{2(1-\lambda_i \mathrm{E}[T'_{h,i}])} \quad = \quad \frac{1+\frac{\lambda_i}{\mu_i \sigma_i}\frac{c^2_{S,i}-1}{2}}{\mu_i \sigma_i - \lambda_i} \tag{4.11}$$

where $\mathrm{E}[T'_{h,i}] = \frac{1}{\mu'_i}$ and $\rho'_i = \lambda_i \mathrm{E}[T'_{h,i}]$ are the mean holding time and the mean presented load of the decomposed *M/G/1* system, respectively. The final bound based on parameters defined in respect to the WFQ system we get from $\mathrm{E}[X^2] = (1+c_X^2)\mathrm{E}[X]^2 = (1+c_X^2)/\mu_X^2$, where the coefficient of variation $c_{S,i}$ of the holding time is evidently the same with both systems. For negative exponentially distributed holding times, $c_{S,i}=1$, we get $\mathrm{E}[T_{f,i}]_{M/M/WFQ} \leq (\mu_i \sigma_i - \lambda_i)^{-1}$.

Noteworthy, the stated upper bound depends on the flows own load contribution only, and not on the loads presented by other flows. This feature is commonly referred to as *flow isolation*. However, for $\lambda_i \geq \mu_i \sigma_i$ the bound becomes infinite because the decomposed system becomes overloaded. This is shown in figure 4.17 for different load splits and weight assignments. Therefore, the bounds yielded by equation 4.11 are in general not sufficient to prove that WFQ outperforms SPQ by granting TDM alike minimum service rates per flow. The tightest bounds, those that prove granted shares, are found for the special case where the load split equals the relative weighting, $\frac{\rho_i}{\varrho} = \frac{g_i}{\sum_j g_j}$.

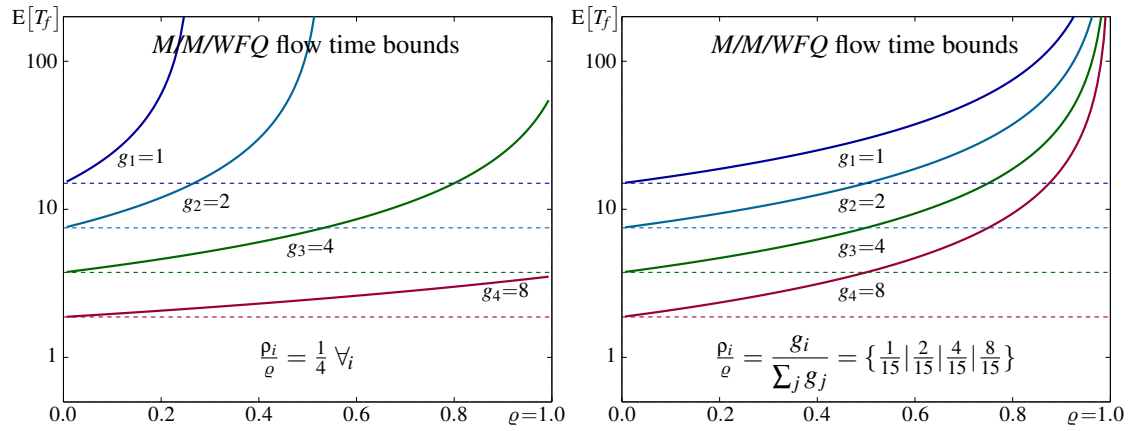


Figure 4.17: Simple WFQ flow time bounds according to equation 4.11 for different load splits and same weights setting, g_i=$\{1,2,4,8\}$, μ_i=1 $\forall_i$, and E$[T'_{h,i}]$ of decomposed *M/M/1* systems (dashed)

Variants and other evaluations of WFQ can be found for example in [88, 102–108]. In [105, lemma 4.1 ff] it is proven that the departure order of the presented WFQ implementation equals that of GPS, as well as the boundedness of the flow time divergence $T_f^{GPS} - T_f^{WFQ} \leq \max_i T_{h,i}$ [105, theorem 4.3] and likewise that of the achieved service shares at any time [105, theorem 4.4]. The integration of a *prioritised flow* in WFQ, for example to excessively privilege signalling, is presented in [104].

Weighted random queue (WRQ)

A more queueing centric approach to weighted processor sharing on a packet-by-packet basis is load aware *weighted random queue* (WRQ+) serving. The switch maintains FIFO queues per flow and serves queues packet-by-packet in a random order that in average achieves the same service sharing as WFQ. In hardware this cannot be realised using shift-registers because the number of required queues is unknown. However, modern switches avoid excessive payload shifting, as shown in figure 4.16 for WFQ. Only the pointers to the memory blocks storing a received payload get queued and exchanged among processing instances. Thus, queues exist in the control memory only, and as required here they can be established dynamically.

To achieve the service shares stated in the left side of equation 4.10, the WRQ+ scheduler needs to serve the first packet of queue i with the state dependent probability:

$$q_i(t) = \frac{g_i x_i(t)}{\sum_j g_j x_j(t)} \tag{4.12}$$

The weighted sum results again quite simply from monitoring arrivals and departures over time, as shown in figure 4.16, and needs not be entirely re-calculated queue-by-queue round-by-round. The major difference to the WFQ scheme is the negligence of the individual packet's size ℓ_{k_a}, defining its holding time $\tau_h(i,k_a) = \frac{\ell_{k_a}}{\mu}$. However, the mean τ_h is assumed to be constant and thus in average we should get the same sharing quotas of WFQ if in average all queues are served with probability q_i.

Thus, we want to know if and to which accuracy does weighted random queue (WRQ+) serving approximate discriminatory processor sharing (DPS). Simulation results for infinte WRQ+ are shown in figure 4.18 together with the analytic curves we got for DPS assuming the same weighting factors (see section 3.3.3). The approximation is good but not perfect because with WRQ+ the *head-of-line* blocking issue per queue is not resolved. On the other side, the FIFO queueing per flow grants *in-order transmission* of flows, a design feature of WRQ that comes without add-on effort. Another feature is that the information how to configure a flow's queue may be derived from a freely definable combination of local parameters, the traffic class identifier within the packet header, deep inspection

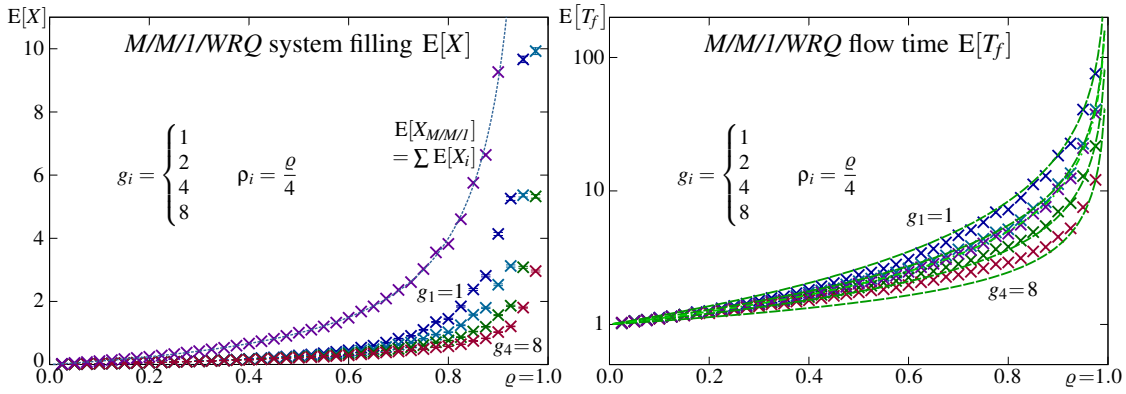


Figure 4.18: WRQ+ performance (simulation results $\times$) compared to DPS (dashed lines) for Poisson arrivals, negative exponentially distributed holding times, and flow weights $g_i = \{1,2,4,8\}$

revealing the service a flow belongs to, and even the current state of the node. However performed, a new flow not necessarily needs to get its own queue. Schemes that integrate class based and flow based queueing can be dynamically configured. Any share assignment and isolation from other flows can be realised and maintained according to current changes of the node's state. As always, the mean performance across all flows cannot be improved, it is dictated by the total load offered. The available mean performance is weight dependently distributed, such that some flows achieve better than average service and others worse.

Turning our attention to more practical finite queueing systems, we first consider a simpler WRQ scheme that implements the lower bound stated on the right in equation 4.10. The queue to serve next is determined by the random selection shown in figure 4.19, based on the presence of flows rather than the number of waiting packets. Normalising the weights g_i of all currently present flows

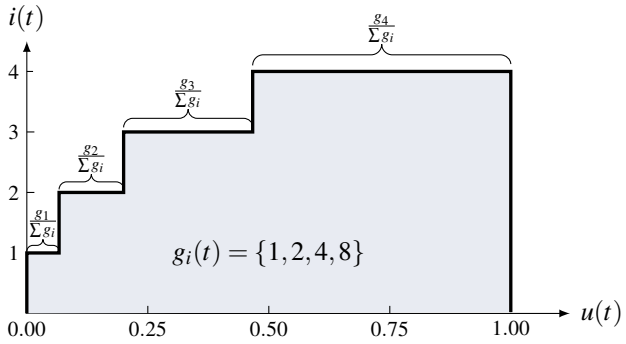


Figure 4.19: WRQ selection based on uniform $u(t)$ and $\frac{g_i}{\sum_j g_j}$ wide intervals per present flow

to $\sum g_i{=}1$ yields the intervals required to select any non-idle queue with probability q_i based on a uniform distributed random number $u(t) \in [0..1]$. The more flows are present, the more steps, and all selection intervals change whenever a new flow appears or a present flow vanishes. However, to evaluate the sharing performance we assume a fixed, non-changing number of flows. Only then we can model the WRQ system by a continuous time Markov chain, as shown in figure 4.20. For presentation clarity the depicted model shows only two flows. Evidently, this can be extended to any number of flows n_f present. However, for rising n_f the continuous time Markov chain quickly becomes intractable. With two flows we get two connected two dimensional plains, with three flows three connected three dimensional spaces, with four flows four connected four dimensional hyper-spaces, and so on. The mechanism is not burdened by this dimensional explosion, it comprises $n_f(t)$ queues and one selection mechanism, its complexity grows linear with $n_f(t)$.

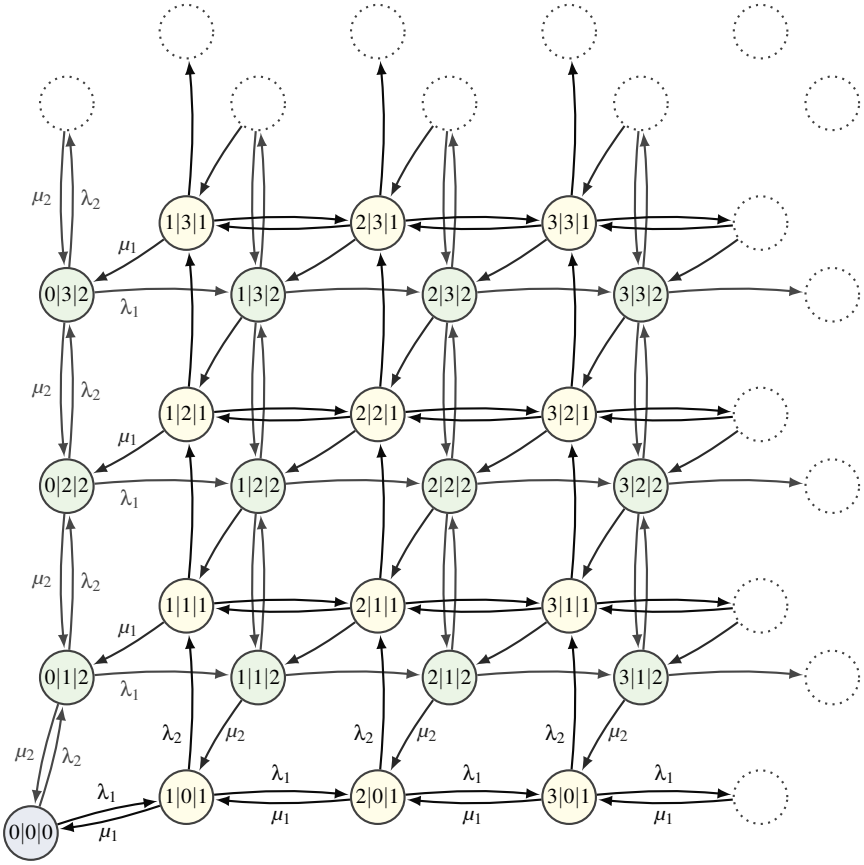
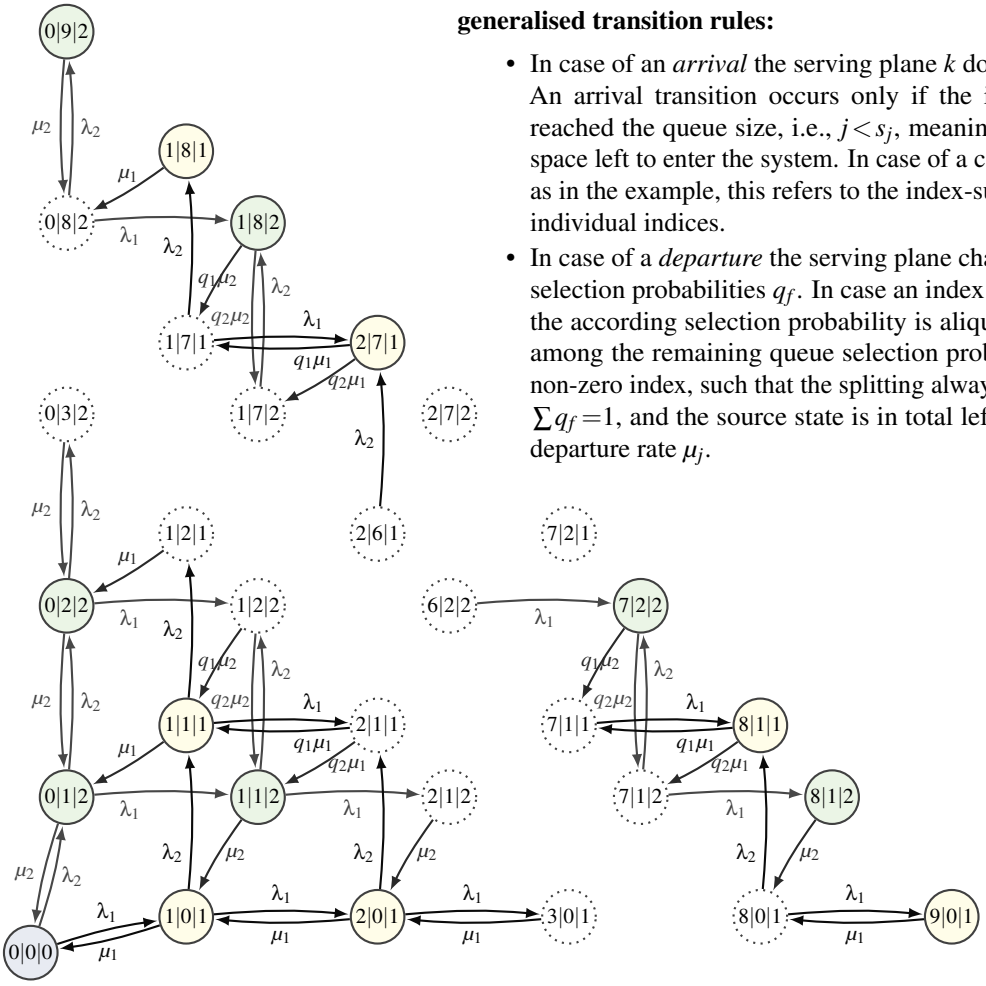


Figure 4.20: WRQ continuous time Markov chain model (*two* flows, *infinite* queues)

Already with the two-flows we cannot identify repeating levels such that we could apply the *matrix geometric method* to solve this infinite system. If we group states such that the total number of customers rises by one per level we get an *increasing number of states, level by level*, and thus, there exist no repeating levels. Grouping into levels where only the number of clients of one flow increases, results in repeating still infeasible levels because of the infinite number of states per level (for which a specific solution approach may exist).

Even for finite queues or a *finite common queueing space*, which better matches reality where the buffering space for payload units is generally upper bounded by the finite memory available, we cannot identify repeating levels with identical internal structure: level by level the *number of states increases with the number of waiting clients*. Thus, the *matrix analytic method* cannot be applied. In consequence, we need to use brute force and solve the potentially huge system of state's *equilibrium equations*. For a realistically chosen queueing space, for example nine times the serving capacity, we get the *finite state transition diagram* shown in figure 4.21, again drawn for two flows only. This is used to grasp the *general transition rules* for states in the centre of any multi-dimensional state transition diagram and how these change close to the edges, stated in the top right of figure 4.21.

Assuming four flows i,j,k,ℓ, we find in the centre of figure 4.22 the state quadruple for identical $i|j|k|\ell$-filling. These states differ only in the currently served queue index. Based on the arbitrarily selected state $(i|j|k|\ell|2)$ all outgoing transitions are shown. Finishing the currently served client causes the departure of it ($j \to j-1$) and serving the next queue with probability q_f. Therefore, we get probability split departure rates $q_f\mu_2$ to the according quadruple with $j-1$ waiting clients of flow 2. These transitions may cause a change in the serving plane. In case of an arrival, the serving plane does not change ($\ldots|2 \to \ldots|2$), only one of the state indices increases ($+1$) with the according arrival rate λ_f. The only exception are arrivals to the idle state, where the arrival causes the server to



generalised transition rules:

- In case of an *arrival* the serving plane k does not change. An arrival transition occurs only if the index has not reached the queue size, i.e., $j < s_j$, meaning that there is space left to enter the system. In case of a common queue, as in the example, this refers to the index-sum rather than individual indices.
- In case of a *departure* the serving plane changes with the selection probabilities q_f. In case an index becomes zero the according selection probability is aliquot distributed among the remaining queue selection probabilities with non-zero index, such that the splitting always sums to one, $\sum q_f = 1$, and the source state is in total left with the full departure rate μ_j.

Figure 4.21: WRQ continuous time Markov chain model (*two* flows, *finite* queues, s_g=9)

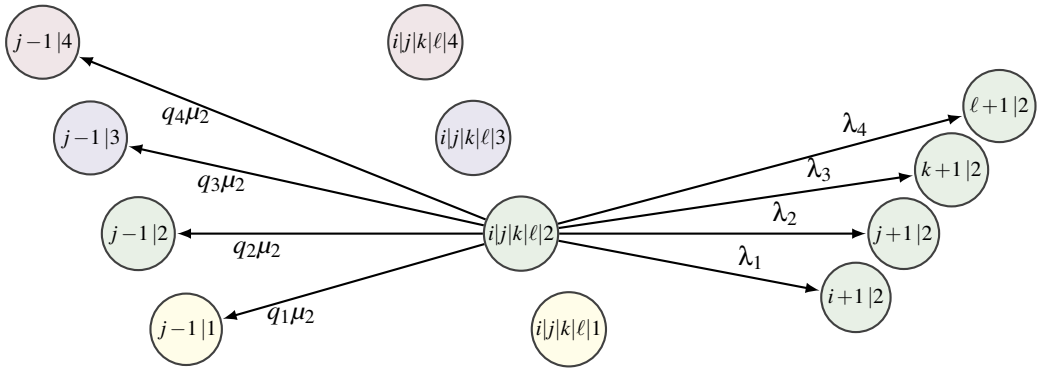


Figure 4.22: WRQ state transitions (*four* flows, apart from any boundary)

instantly start serving the arriving customer ($\ldots|0 \to \ldots|j$). Note that no transition out of any other state within the quadruple shown in figure 4.22 leads to a state shown in the figure.

Using the per state departure transitions depicted in figure 4.22 and the rules stated with figure 4.21 we can numerically fill the *state transition matrix* Q and solve the equilibrium equations system to get all state probabilities $p_{i|j|k|\ell|s}$.

Having calculated the state probabilities p by solving $Q\,p = 0$ for $\sum p = 1$ we get the system fillings $\mathrm{E}[X_i]$, the blocking probabilities $P_{b,i}$, and other mean system properties, either directly

create, fill, and solve state equilibrium equations $Q\,p = 0$

1. create a list of all state-index permutations
2. remove all impossible state-index combinations
3. initialise a square Q-matrix with size = length of state-index list
4. use the indices of the state-index list as Q-matrix indices
5. for every possible state-index combination in the reduced list (Q-matrix row)
   (a) get the Q-matrix column of possible successive states (according to figure 4.22)
   (b) insert according transition rates in Q-matrix (following the rules stated in figure 4.21)
6. if $\exists$ empty rows remove them and their associate columns
7. insert the negative Q-matrix row sums as diagonal elements
8. replace the last column of the Q-matrix by a *ones*-vector
9. create a *zeros*-vector b and replace the last entry by *one*
10. solve the linear equation system $Q\,p = b$ to get the state probabilities p

or indirectly from each other, using the common obvious calculation rules, and applying Little's law $N = \lambda T$ to get *sojourn times* from presence numbers.

calculate mean system properties

1.	$Q\,p = 0,\ \sum p_{ij\ldots q}{=}1$	$\rightarrow\ p_{ij\ldots q}$	solve equation system $\rightarrow$ state probabilities
2.	$\mathrm{E}[X_i] = \sum i\,p_{ij\ldots q},$	$\mathrm{E}[X] = \sum \mathrm{E}[X_i]$	presence weighted sums $\rightarrow$ system fillings
3.	$\mathrm{E}[Q_i] = \sum (i{-}\xi_i)\,p_{ij\ldots q},$	$\mathrm{E}[Q] = \sum \mathrm{E}[Q_i]$	waiting weighted sums $\rightarrow$ waiting clients
4.	$P_{b,i} = \sum p_{ij\ldots q}(i{=}s_i),$	$P_b = \sum \frac{\lambda_i}{\lambda} P_{b,i}$	q_i full $\hat{=}\,\nexists\,\lambda_i$-exit $\rightarrow$ blocking probabilities
5.	$\vartheta_i = \mu_{i,M} \sum p_{ij\ldots q}(q{=}i),$	$\vartheta = \sum \vartheta_i$	$\mu_{i,M}\times$ serving states sum $\rightarrow$ throughputs
6.	$\mathrm{E}[T_{f,i}] = \mathrm{E}[X_i]\,/\vartheta_i,$	$\mathrm{E}[T_f] = \sum \frac{\vartheta_i}{\vartheta} \mathrm{E}[T_{f,i}]$	use Little's law $N = \lambda T$ $\rightarrow$ flow times
7.	$\mathrm{E}[T_{w,i}] = \mathrm{E}[Q_i]\,/\vartheta_i,$	$\mathrm{E}[T_w] = \sum \frac{\vartheta_i}{\vartheta} \mathrm{E}[T_{w,i}]$	use Little's law $N = \lambda T$ $\rightarrow$ waiting times

The *octave* code performing all this, for any number of flows and shared as well as individual queues of any size, is presented in addendum A.I.3. However, we need to admit that even though we use a sparse Q-matrix, *octave* fails to solve the equation system if the number of existing states reaches several thousands. The example studied, with four flows and a shared queue of size s=9, causes 1981 existing states, for which *octave* showed no infirmity.

The continuous time Markov chain precisely models the finite queueing system, and thus, solving the equilibrium equations that result from the state transition diagram yields results that exactly match the simulation results, as shown in figure 4.23. On a first glance, the identical, weight independent *throughput* ϑ is a surprise. This is not the behaviour we expect from a discriminatory scheduling discipline. However, if we reconsider the continuous time Markov chain representing this shared queue system and look at the blocking states, being the diagonal that binds the system filling, we recognise that the states along this line constitute the blocking states for all flows. Thus, the blocking probability is the same for all flows, and in consequence are in case of equal load shares also the throughputs ϑ_i identical. Looking at the system filling $\mathrm{E}[X_i]$ and the waiting times $\mathrm{E}[T_{w,i}]$ we recognise that these are not weighting proportionally spaced, and that they diverge in heavy overload. This indicates that the highly weighted flows are quicker served than they arrive, whereas the lowest weighted flow is served so rarely that in overload the queue is heavily occupied by rarely served low priority packets. This is not comparable with *discriminatory processor sharing* (DPS).

To better approximate the beneficial DPS features we change back to the original queue selection stated in equation 4.12, where the number of present clients is considered. Precisely, we select the

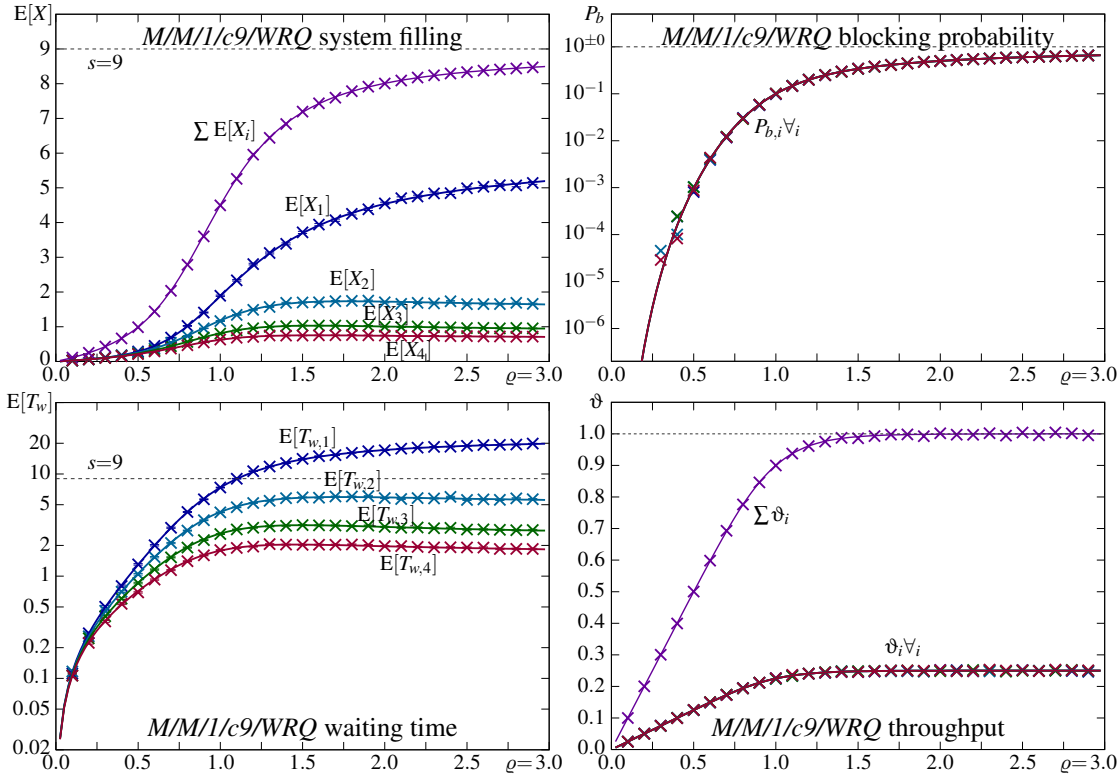


Figure 4.23: WRQ performance (simulation: $\times$, calculation: solid lines), shared queue with size s=9, Poisson arrivals, negative exponential T_h, equal load shares $\rho_i=\frac{\varrho}{4}$, and weights $g_i=\{1,2,4,8\}$

next queue to serve with the probabilities

$$q_i = \frac{w_i\, g_i}{\sum_j w_j\, g_j} \tag{4.13}$$

where w_j is the number of waiting clients, which equals the present clients x_j after the currently served has departed. Using the new splitting factors in figure 4.22 and the continuous time Markov chain depicted in figure 4.21 we get the model for the WRQ+ scheme. The results shown in figure 4.24 indicate a slightly lesser discrimination of flows. At overload the shared queue is less occupied by low priority packets and the negative slope for increasing system load also vanished, here shown for the flow time $\mathrm{E}[T_f]$. The blocking probability and the throughputs remain flow independent. The absence of any starvation tendency and the nearly even separation of the $\mathrm{E}[T_f]$ curves conforms the native expectation on a fair weighting scheme.

Next we want to see what happens if we provide individual queues per flow. These will be less effectively utilised, so for an approximately comparable system we set the queue sizes $s_i=3$ per flow. The analytic results from solving the state transition diagram, set up as before just without cutting off the upper right triangle, and simulation results are shown in figure 4.25. As could be expected, with individual queues the equal $P_{b,i}$ vanishes and also the throughput becomes quite different. We actually recognise a *throughput drop*, even for the WRQ+ where we consider the number of waiting customers as stated in equation 4.13. This occurs because here the throughput reduction results from the increased blocking probability $P_{b,low}$, which is a result of the higher queue filling $\mathrm{E}[X_{low}]$. However, the throughout ϑ_{low} does not drop to zero, with WRQ+ less than with plain WRQ. The weight distribution conform spacing among both, the waiting times $\mathrm{E}[T_{w,i}]$ and the throughputs ϑ_i, is consistent with the expected behaviour. Thus, WRQ+ with individual queues is a *weighted fair queueing* policy that shares the resources such that both prime metrics, the *sojourn times* and the *loss rates*, are distributed according to the weights assigned. In consequence are also the *throughputs*

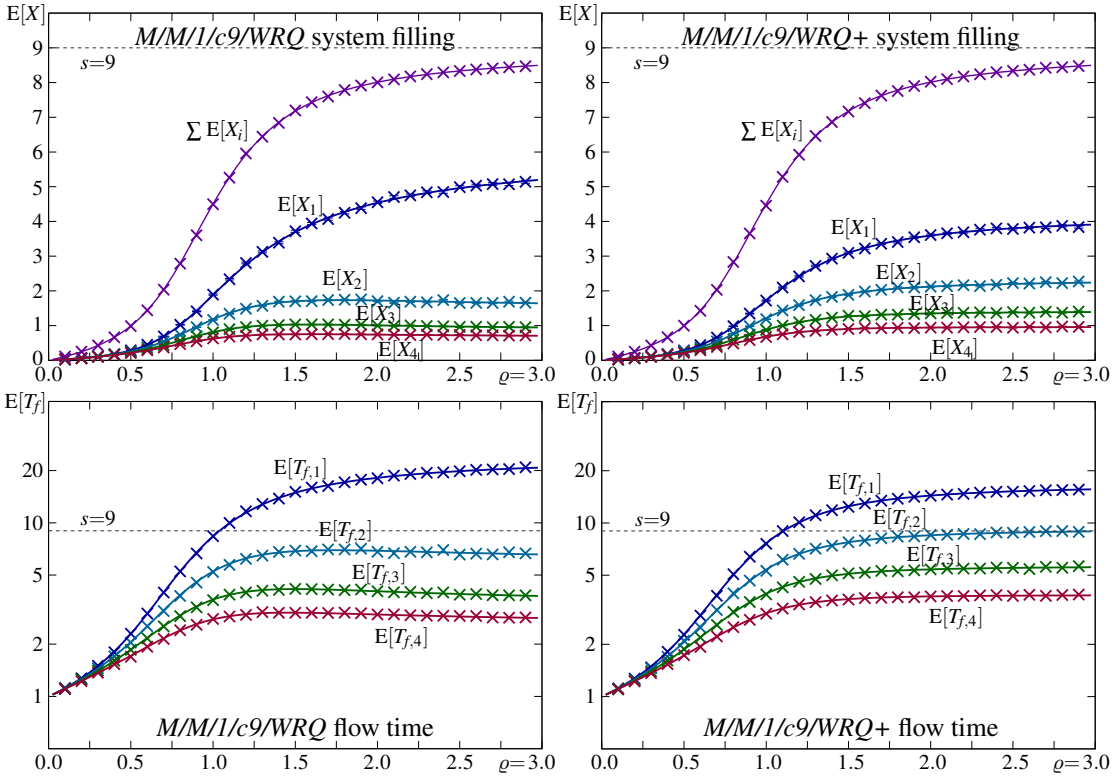


Figure 4.24: WRQ (left) WRQ+ (right) performance (simulation: ×, calculation: solid lines) shared queue s=9, Poisson arrivals, negative exponential T_h, equal load shares $\rho_i=\frac{\varrho}{4}$, weights $g_i=\{1,2,4,8\}$

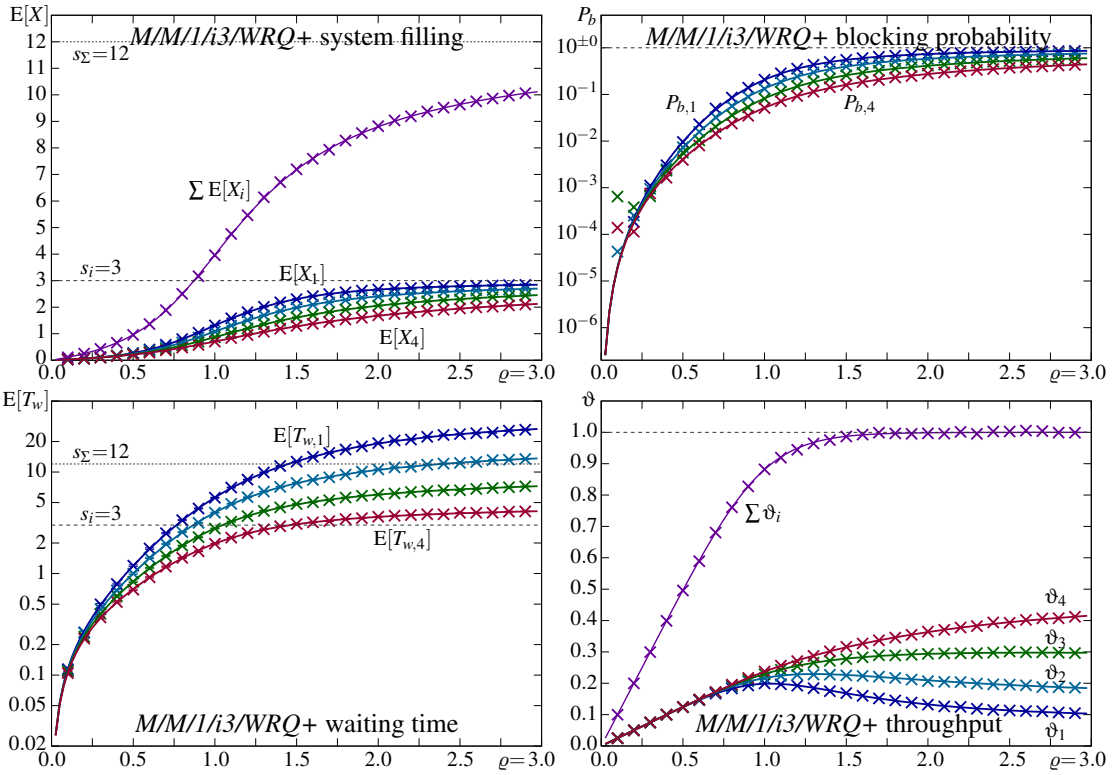


Figure 4.25: WRQ+ performance for individual queues per flow (simulation: ×, calculation: solid lines) Poisson arrivals, negative exponential T_h, equal load shares $\rho_i=\frac{\varrho}{4}$, weights $g_i=\{1,2,4,8\}$

distributed, though not exactly as the weights are (nearly linear distributed although logarithmic weights were assigned).

An interesting side effect of WRQ with individula queues, and even more with WRQ+, is a slight smoothing experienced by the departure flows of higher weighted flows. In figure 4.26 the monitored coefficient of variation of arriving (+) and departing (×) flows are shown for the different WRQ variants evaluated. That the departure flow of the least privileged flow shows a little burstiness

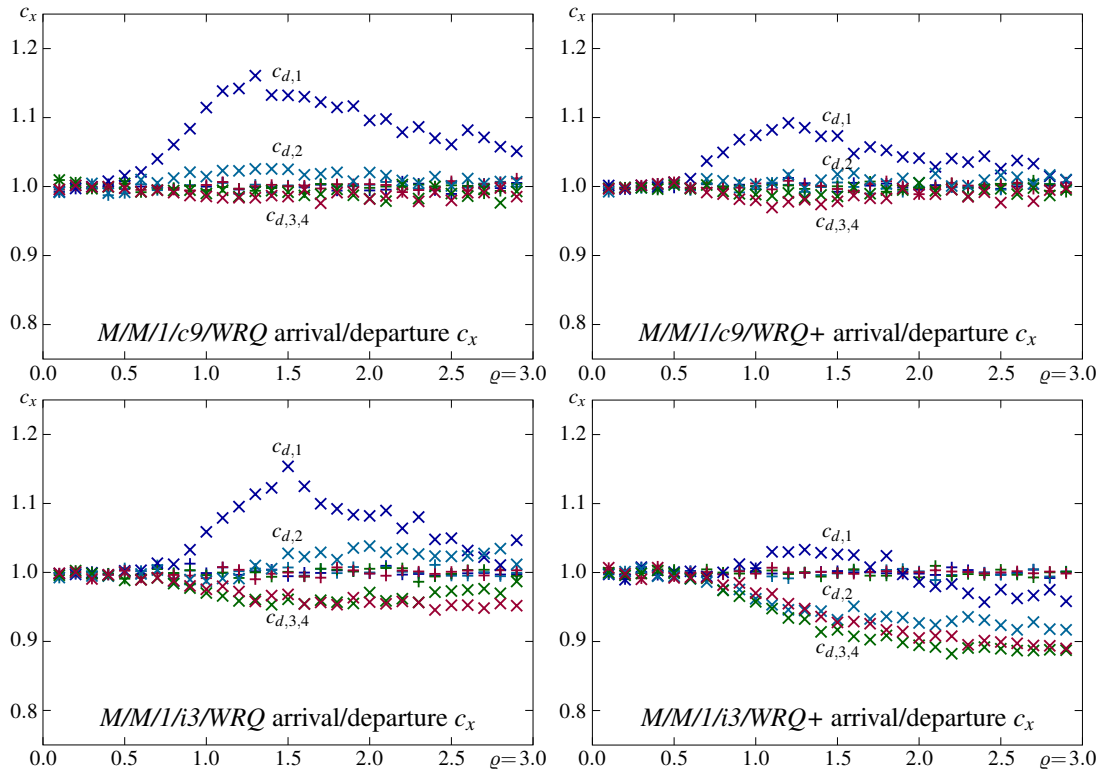


Figure 4.26: Effect of WRQ variants on the departure flow's coefficient of variation, Poisson arrivals, negative exponential holding times T_h, equal load shares $\rho_i = \frac{\varrho}{4}$, weights $g_i = \{1,2,4,8\}$

is expectable, it is less frequently served than the others, and also that this effect vanishes the more the system is overloaded (equalisation by randomisation). But that in case of individual queues (lower two figures) the higher privileged flows show a departure process that is less variant than both, the arrival and the service process, is not so evident. The finite queue and the non-preemptive serving of less prioritised flows seems to cause this positive effect. However, to prove this we would need to solve the differential equations representing the continuous time Markov chain, which for such a huge state transition diagram is likely not possible, given that solving the linear equations system in order to get the steady state probabilities already challenges *octave*. That the means fit is not sufficient to prove that the simulation is correct as well concerning higher moments. Likely it is, and therefore the results are shown without proof, concluding the WRQ discussion.

Weighted round robin (WRR)

In figure 4.27 flow based *weighted round robin* (WRR) queue serving is depicted. As with WRQ a dedicated queue is maintained in the control logic for every flow present. The ellipsis indicates that this causes a varying number of queues that dynamically changes as flows arrive and depart.

Achieving the service probability q_i bound stated in equation 4.12 by a *round robin* approach rises some practical challenges. Firstly, the number of queues required and visited per round is dynamic because a queue exists only while a packet from the according flow i is awaiting service.

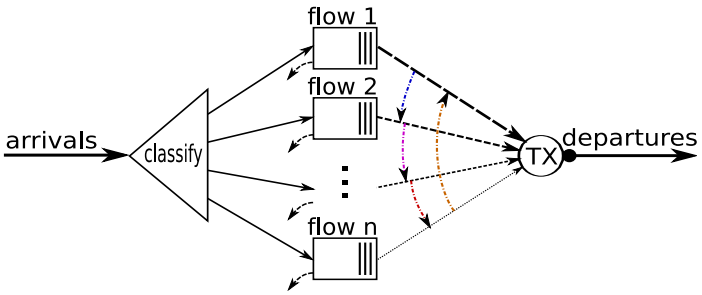


Figure 4.27: Weighted round robin – WRR

Secondly, as with customs based queueing, either $c\,q_i(t)$ packets need to be served per visit, or the queue has to be visited $c\,q_i(t)$ times per round, serving one packet per visit. The multiplier c is to be chosen such that for all present $q_i(t)$ the product $c\,q_i(t)$ yields integer numbers $\forall_i$ at any time t. This value c can be huge and may change during a round, and is therefore hardly manageable. A scheme that considers the number of waiting packets per queue is therefore rather utopian.

To evaluate WRR we implement a rather simple algorithm that achieves a serving schedule per round that is somewhere in between serving $c\,q_i(t)$ packets per visit and $c\,q_i(t)$ visits per round: assuming that all weights g_i are defined as integer numbers, we schedule one queue after the other, every time decrementing the weight by one, g_i--, and skipping queues with zero weight. When all g_i have become zero a round is finished and the next starts with updated weights vector $\vec{g}(t)$. In *octave* this scheme is implemented by

simple weighted round robin schedule

```
for r = 0 : max_i(g) - 1
    gr = find(g.-r > 0);          get indices i of weights g_i > r
    rrs = [rrs;gr];               append indices to previously found
endfor
```

where we actually do not decrement the weights, still achieve the same resultant schedule *rrs* per round. Note that this scheme defines the visits assuming that one packet is served per visit. However, in general we get blocks in the sequence where the same queue is repeatedly visited, such that $g_i - g_j$ packets are served before some other queue is served. These repeated visits are not optimally distributed across a round. However, this has no impact on the mean performance because only the higher moments depend on the visit distribution within rounds.

For a given snapshot, assuming a fixed number of flows present, we might try to represent WRR by a continuous time Markov chain model. To do so we would need to add one more dimension per flow to the continuous time Markov chain shown in figure 4.21 in order to consider repeatedly serving the same queue. Concerning mathematical treatment we need to note that the WRR serving cycle introduces auto-correlation. The number of queues served in between one visit and the next is precisely defined, and if no new queue is invoked or an idle queue removed, the perfectly same serving order is repeated round by round.

Simulation is the straightforward tool of choice to evaluate how good this simple WRR scheme approximates DPS. In figure 4.28 we show simulation results for WRR and analytic results for DPS (dashed lines) using the same weighting factors $g_i = \{1,2,4,8\}$ for the four flows assumed present. The total system filling $\sum \mathrm{E}[X_i]$ and the average flow time $\sum \lambda_i \mathrm{E}[T_{f_i}]$ are the same as for *M/M/1*. This is evident, and shows that the simulation is not completely faulty. Comparing the mean flow time $\mathrm{E}[T_{f_i}]$ per flow, the samples shown in the right graph of figure 4.28, with the analytic results for DPS, the dashed lines in the same graph, calculated using equation 3.111 presented in section 3.3.3, we recognise that this WRR realisation does not approximate DPS well. At low loads the differentiation achieved with WRR is less than with DPS, and at high loads WRR is too aggressive.

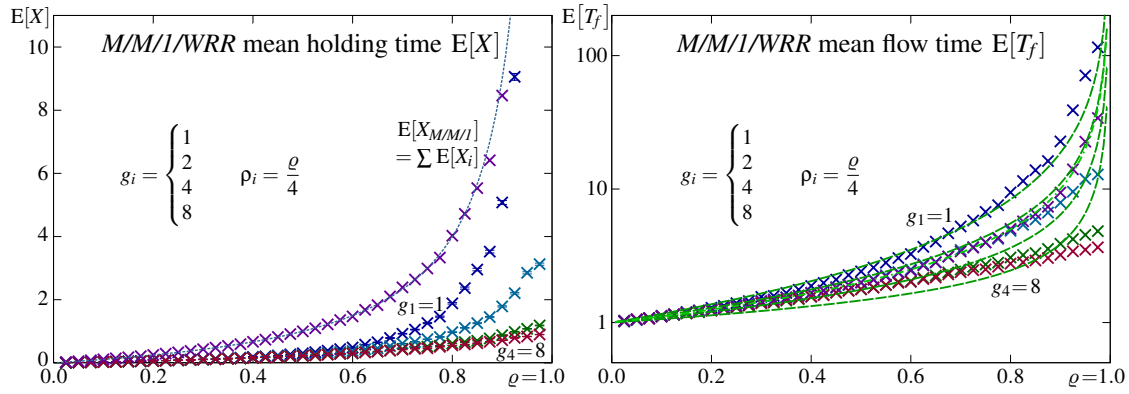


Figure 4.28: WRR performance (simulation results) compared to DPS (dashed lines) for Poisson arrivals, negative exponentially distributed holding times T_h, and weights $g_i = \{1,2,4,8\}$

The reason resides in the implementation of equation 4.10, which actually implements the lower bound, as already used with plain WRQ and depicted in figure 4.19. The clear divergence that results from this seemingly minor change highlights the *importance of state awareness*. If we consider the weights g_i but not the number of present clients per weight x_i, than we do not achieve a differentiation comparable to the ideal, fluid presuming, discriminatory processor sharing.

Figure 4.29 presents simulation results for a finite WRR system using the algorithm presented above and numeric results for plain WRQ with individual queues ($s = 3$). The numeric results

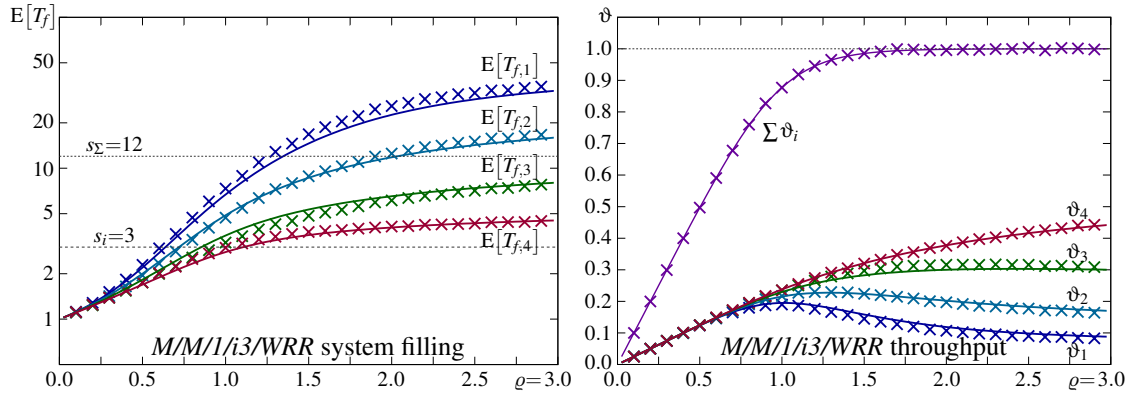


Figure 4.29: Finite individual queues WRR (simulation results) compared to plain WRQ (solid lines), individual queue, $s_i = 3$, Poisson arrivals, negative exponential T_h, and weights $g_i = \{1,2,4,8\}$

approximately match the first moments, as could be expected because in average they do not depend on the service cycle, if both schemes serve the individual queues equally often. However, we do recognise slight deviation for less weighted flows. Thus, even if higher moments are not of interest, the plain WRQ state transition diagram only approxiamtely reflects the behaviour of WRR. How the cycles are defined may differ. In [109] we find for example a *deficit counter*, which is incremented by a weight dependent quantum whenever a server *visits* a non-idle queue, and decremented by the packet size for each packet served during the visit. If this counter would become negative or the queue became idle, the server *moves* to the next queue. An unused *deficit* is carried over to the next visit. This implementation considers the packet size. In average this should be equal to a bounded number of packets per round. However, the results presented in figure 4.29 show that such minor differences in the cycle definition can influence the performance.

Next we change to shared spaces, and compare in figure 4.30 the numeric results for a somewhat artificial WRR system with common queuing space, $s = 9$, approximated by plain WRQ, with

simulation results for a bounded DPS where we add a corresponding upper-bound on the number of clients that can be served in parallel, $n_{max} = 9$. As could be expected, the performance of WRR,

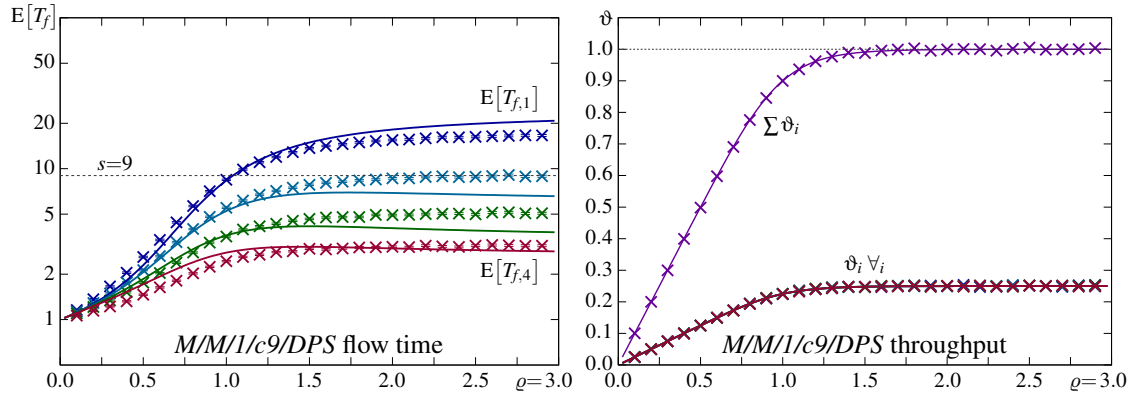


Figure 4.30: Bound DPS (simulation results) compared to plain WRQ (solid lines), sharing bound $s, n_{max} = 9$, Poisson arrivals, negative exponential T_h, and weights $g_i = \{1,2,4,8\}$

here approximated by plain WRQ, is worse than that of the ideal upper bounded DPS system, which shows no decline. The blocking probabilities $P_{b,i}$ and throughputs ϑ_i match, but these are determined by the shared space and not separated by the flow weighting. That a bound on the shared queue and a bound on the server sharing yield comparable performance proves the equivalence of these systems. The bounded variant of DPS, which we mentioned but not analysed in the end of section 3.3.3, could be used as benchmark for any finite weighting system that employs a shared queueing space. However, benchmarks that rely on simulation are not the most efficient option.

Class based queueing (CBQ)

To conclude this section on weighting based scheduling we finally consider different queue sizes. This causes a queueing and scheduling scheme that resembles the so called *class based queueing* (CBQ), sketched in figure 4.31 and sometimes referred to as *customs based queueing* to separate it from other classification based mechanisms. In contrast to WFQ and its derivatives WRQ and WRR,

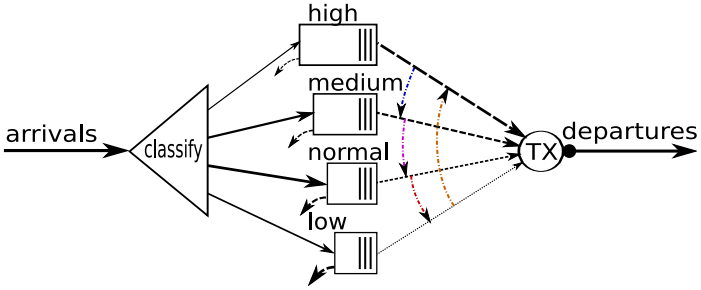


Figure 4.31: Customs based queueing – CBQ

is CBQ commonly used with traffic classification, meaning that queues are provided and maintained per traffic class, not per flow, and all flows assigned to the same traffic class share the same queue. This reduces the control complexity and enables hardware implementation via shift registers because the number of queues is given by design, independent of present traffic flows.

The queues are usually served in *round robin* fashion, and in addition to the different queue sizes provided, differentiation is achieved by upper bounds on the number of packets served per visit (*weights*). This introduces memory into the serving process, such that we cannot model it precisely by the simple continuous time Markov chain shown in figure 4.21. We again would need to add one more dimension per flow to cover this, making the state transition diagram quite complex and due to

the excessive number of states numerically problematic due to the consequentially many more close to zero state probabilities.

For an approximate numeric evaluation we therefore assume random queue selection as with plain WRQ. This should perform similarly, at least concerning the first moments, and because presence awareness is for CBQ not applicable. To disclose this difference we hence use the abbreviation CBRQ, inserting the term 'random'. The performance of this scheme is shown in figure 4.32. Evidently, model and simulation fit each other because we use the same CBRQ scheme for both:

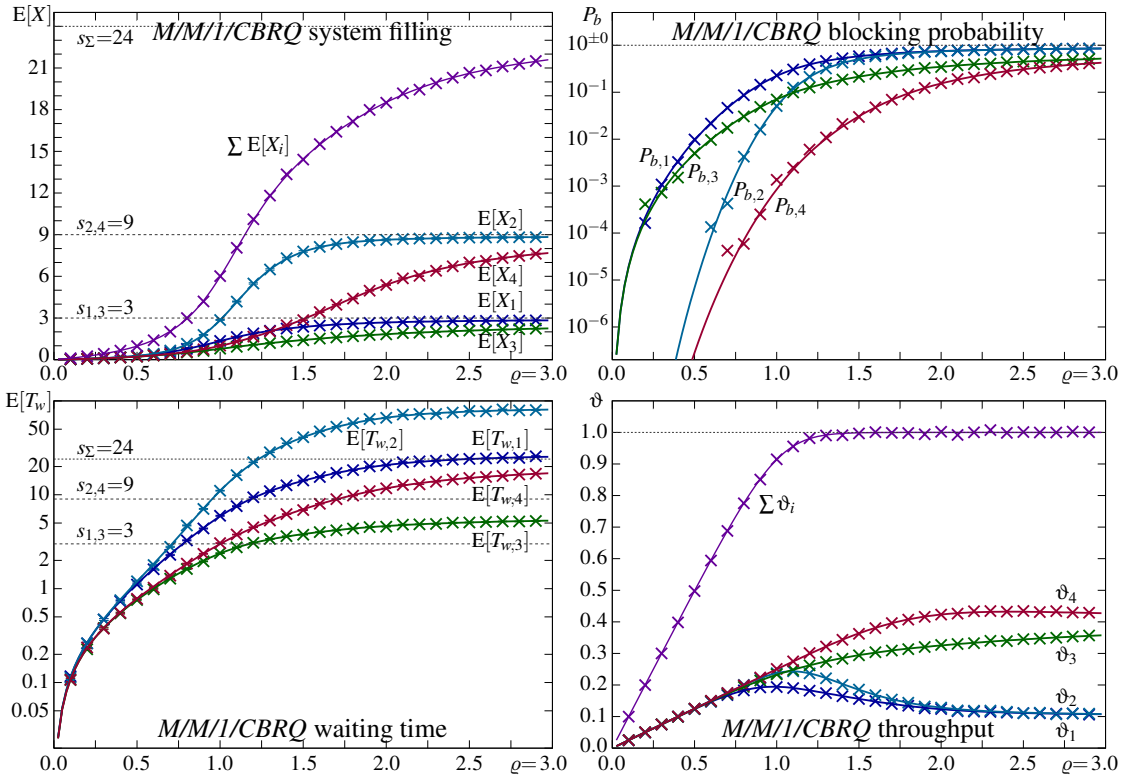


Figure 4.32: CBRQ performance (simulation: $\times$, calculation: solid lines) for $s_i=\{3,9,3,9\}$ per traffic class, Poisson arrivals, negative exponential T_h, equal load shares $\rho_i=\frac{\varrho}{4}$, and weights $g_i=\{1,1,4,4\}$

with simulation the queue selection according to figure 4.19, and the model is an adjusted state transition diagram according to figure 4.21. Comparing CBRQ with WRQ+ and WRR, where the individual queues were equally sized, we recognise that with CBRQ the flow-time $\mathrm{E}[T_f]$ and the loss performance P_b can be individually addressed. Expectedly, a bigger queue reduces the loss probability at the price of higher flow times. Higher prioritisation achieves a similar differentiation as before, but only for flows with equally sized queues. With CBRQ the losses and flow-times are not necessarily distributed according to the weights, only the ordering of the throughputs follows the weights order (for equal ingress load shares).

Response to non-Markovian arrival and service processes

The real world inter-arrival times of packets tend to be more bursty than negative exponentially distributed, and on the other side are the holding times in general lower bounded by the minimal packet size (essentially the header size) and upper bound by some standardised limit, necessary to suitably dimension processing buffers. In section 2.2 we introduced random processes capable to stochastically represent such traffic characteristics. The *Lomax distribution*, also known as *Pareto type II*, can be used to model an inter-arrival time distribution with infinite variance. To reveal the effect of unpredictable arrivals we choose this, in particular $L(2,1)$, the boundary to infinite variance.

For the service process we choose the *Beta distribution*, and from the many possible shapes a tilted bath-tube shape with peaks at the lower and upper bound $\mathscr{B}(\frac{1}{3},\frac{2}{3},\frac{1}{2},2)$. This covers many short messages (ping) and more huge packets (pong) than medium sized. The mean holding time remains one, $\mathrm{E}[T_h] = \frac{1}{2} + \frac{3}{2}\frac{1}{3}/(\frac{1}{3}+\frac{2}{3}) = 1$. The upper bound is four times the lower bound, narrow if we compare it with IP-packets where the relation is in the thousands. However, if packet-size independent latencies exist, for example a 10 *ms* sync-time, than the holding time variance shrinks drastically. If the latency exceeds the maximal packet size, the relation among lower and upper bound drops below two, and the service process becomes more and more deterministic.

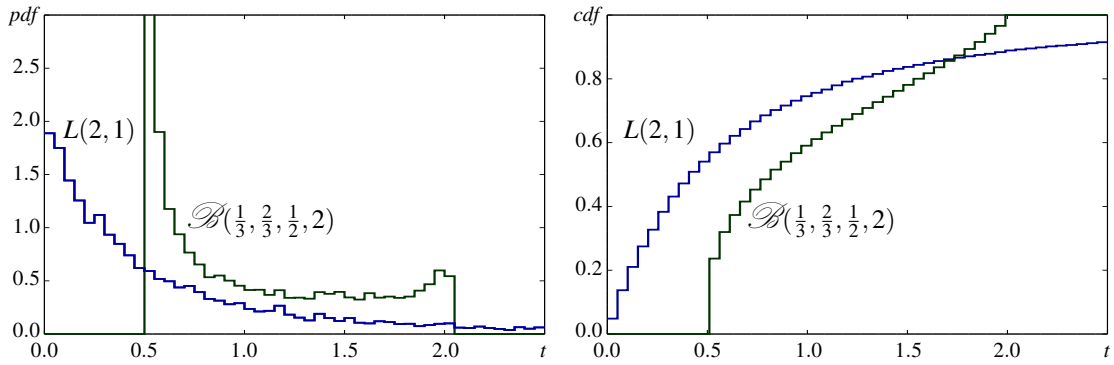


Figure 4.33: Arrival and service process histograms of evaluated $L/\mathscr{B}/1/s_i/\ldots$ systems

Using these arrival and service processes, shown in figure 4.33, we repeat the simulations for the different weighting realisations discussed so far, and show the results together with the analytic results we get for negative exponentially distributed arrival and service times.

Figure 4.34 shows the results we get for *CBRQ*. We recognise that the high variability of the

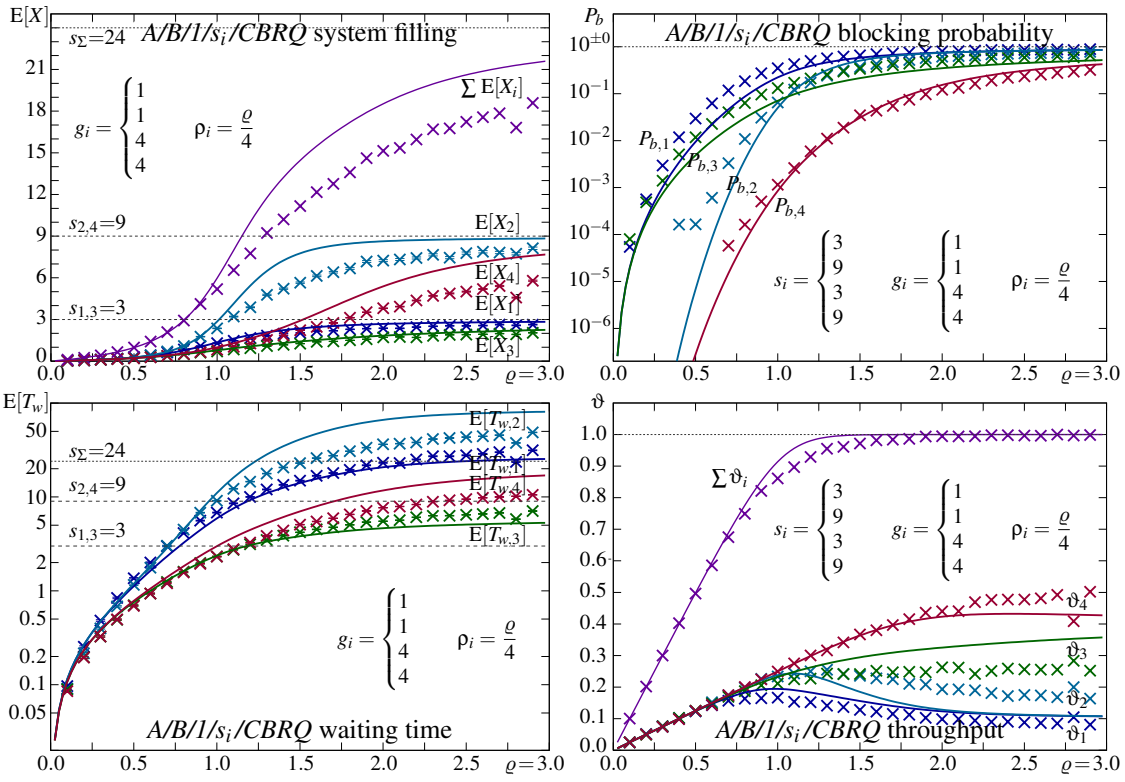


Figure 4.34: $L/\mathscr{B}/1/s_i/CBRQ$ simulation ($\times$) compared to $M/M/1/s_i/CBRQ$ calculation (solid lines)

arrivals causes a less effective queueing system (lower total queue filling). Actually, in overload the

longer queues are in average considerably less filled, and more arrivals are blocked. This heavily affects the service differentiation introduced by different queue sizes: the flow time discrimination shrinks, whereas the throughput discrimination increases. The discrimination caused by different weights remains approximately the same.

Figure 4.35 shows the mean flow time $\mathrm{E}[T_f]$ and the throughput ϑ for *WRR* with equally sized queues of size $s_i = 3 = 12\frac{\lambda_i}{\varrho\mu}$. We recognise that the less effective queue filling results in decreased

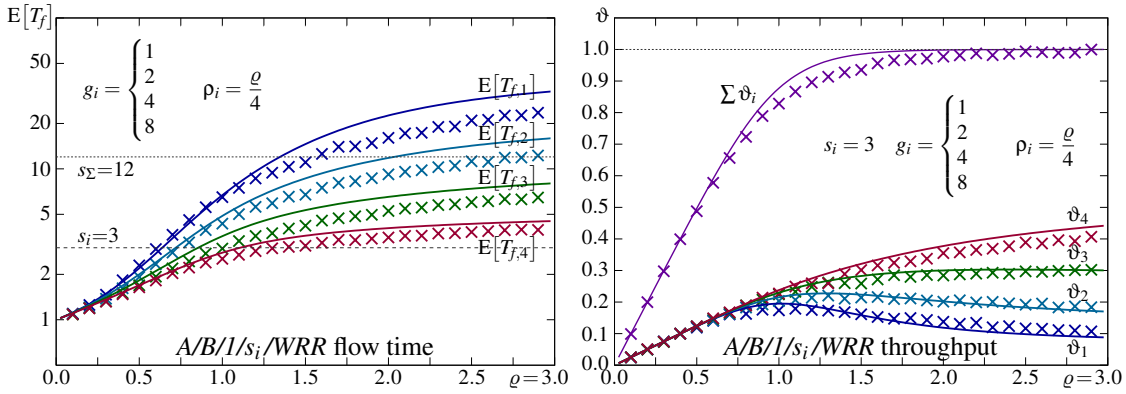


Figure 4.35: *L/$\mathscr{B}$/1/s_i/WRR* simulation ($\times$) compared to *M/M/1/s_i/WRQ* calculation (solid lines)

flow times, now for all flows, and reduced flow discrimination in terms of throughput shares. Thus, to compensate bursty arrivals the weights need to be adjusted. However, this is a complex task because every change of any g_i affects the performance of all flows.

Switching to a shared queue, and applying the initially developed WRQ+ scheme, figure 4.36 shows the system filling $\mathrm{E}[X]$ and the flow time $\mathrm{E}[T_f]$ we get for $s = 9 = 9\frac{1}{\mu}$. We recognise that the

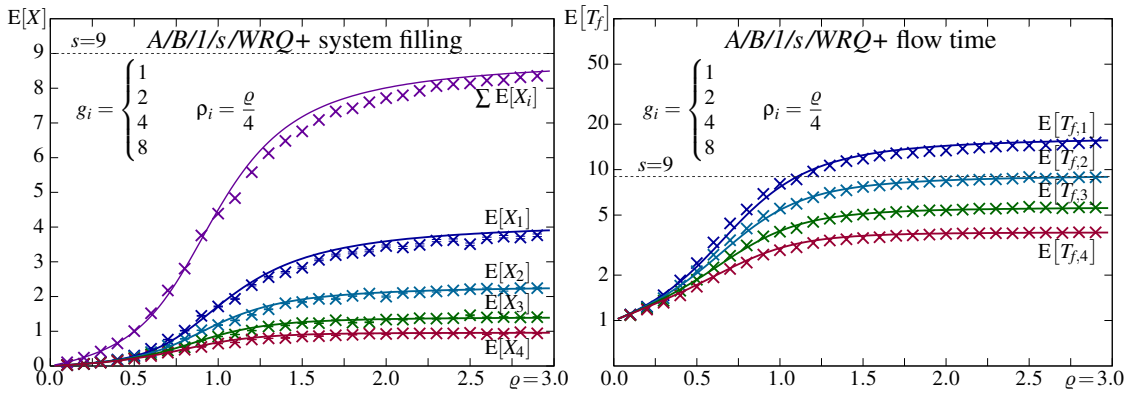


Figure 4.36: *L/$\mathscr{B}$/1/s/WRQ+* simulation ($\times$) compared to *M/M/1/s/WRQ+* calculation (solid lines)

impact of the heavy arrival process nearly vanished. The deviation of the simulation results ($\times$) for *L/$\mathscr{B}$/1/9/WRQ+* from the analytic results (solid lines) representing *M/M/1/9/WRQ+* is marginal. Only for the least weighted service we recognise a slight reduction of the queue filling and in consequence slightly shorter flow times.

This changes completely when we consider infinite queues. With infinite queues the excessive load peaks caused by bursts are not repelled via blocking them off due being full. Figure 4.37 shows the results for *WRQ+* with infinite queues. If the queue is shared or not, makes no difference. The bursty arrivals cause that the mean queue filling is increased, and thus, also the mean flow times are increased. Such bursty arrivals have a very negative effect on the mean response time of infinite systems, compared to finite systems with a total queueing space $s \sim 10\frac{1}{\mu}$, where the opposite occurs at the price of increased blocking.

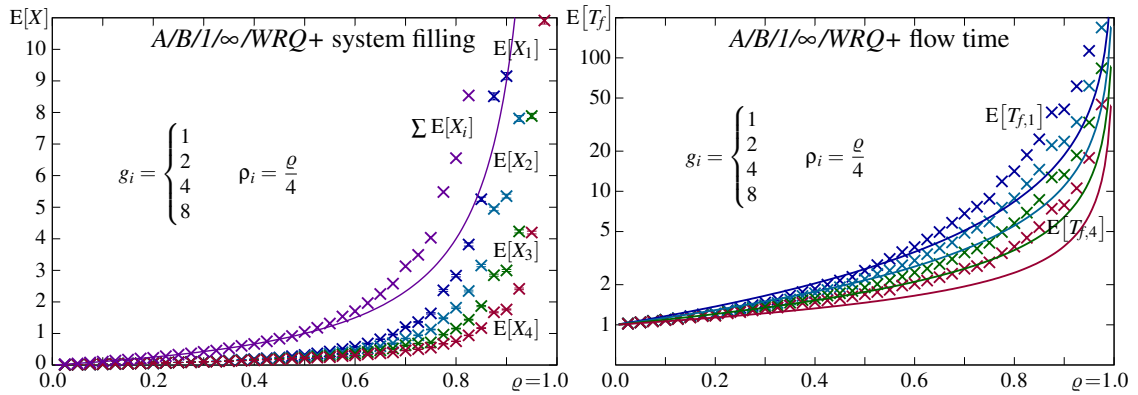


Figure 4.37: *L/ℬ/1/∞/WRQ+* simulation (×) compared to *M/M/1/∞/DPS* calculation (solid lines)

Studying non-Markovian processes we evidently are also interested in the effect the queueing system has on the departure characteristics. In figure 4.38 we compare the monitored arrival and departure coefficient of variation, one by one, and among different weighting schemes. As reference

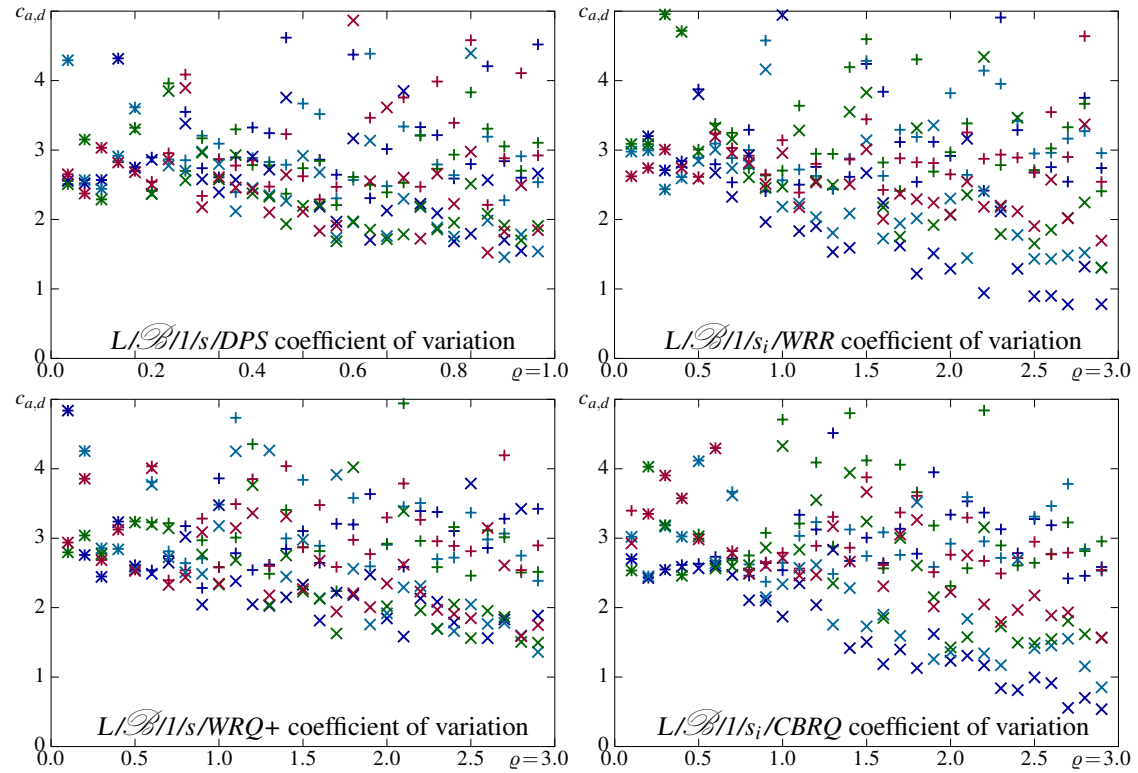


Figure 4.38: Monitored coefficient of variation of arrivals (+) and departures (×)

we choose bounded *discriminatory processor sharing* (DPS), as introduced above with WRR. At low loads equals the departure coefficient of variation that of the arrivals. In heavy overload the departures are considerably smoothed, but the chaos introduced by the arrival process with its infinite variance in not removed. The monitored departure coefficient of variation fluctuates as it does for the arrivals. Weights related smoothing is only recognisable for WRR and CBRQ, the more the less a flow is weighted. If we reconsider the coefficient of variation results found for their *M/M/1/s/XXX* counterparts, we may conclude that latter again results form the averaging by randomisation. That for DPS and WRQ+ no weights dependence is visible supports the assumption that WRQ+ truly approximates DPS independent of the involved random processes.

Some final remarks on weighted scheduling

The lower bound stated in equation 4.10 is rather weak because the number of flows summed in the denominator is not limited. Thus, the granted service shares can approach zero, $\sigma_i \to 0$, at least in principle, if the accepted load approaches the server capacity, $\sum_i \lambda_i^{in}/\mu \to 1$. Theoretically, the least weighted traffic could cause saturation for all service classes. Luckily, infinite packet queues are not feasible and practical overload is restrictively bounded by the finite capacity of communication network node's ingress ports.

A node with n symmetric ports, as sketched in figure 4.39, where per port the ingress load is limited by the line-rate, $\lambda_{\max}^{in}=\mu$, can at maximum cause a persistent ingress load of n times the line-rate μ. Accordingly is the sum in the denominator of equation 4.10 upper bounded by $n\,g_{i,\max}$.

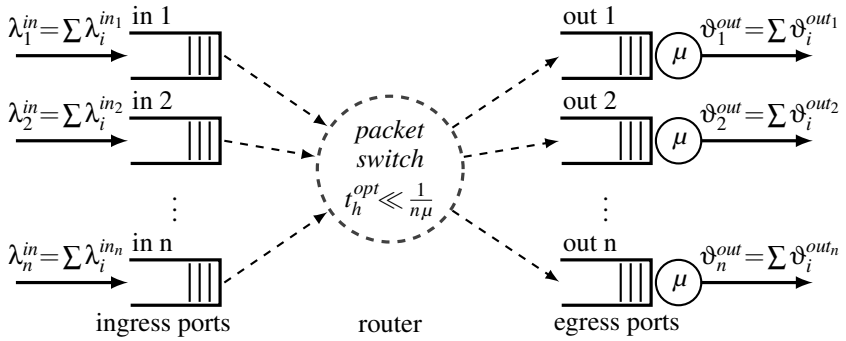


Figure 4.39: Symmetric network node, $\lambda_{\max}^{in}=\mu \Rightarrow \sum\lambda^{in}(t) \leq n\mu$

Thus, in practice weighted queueing that implements equation 4.10 does grant finite minimum performance per flow, and the minima strictly reflect the weights assigned to all present flows. If in addition the mean traffic volumes per class are limited across all ingress ports to less than the egress capacity, $\sum_k \lambda_i^{in_k}(t) \leq \mu$, prior being routed to an egress port, this lower bound can be considerably improved, even though the worst case sum over all classes still exceeds the egress capacity. This clearly shows how superior weighted scheduling is compared to crude prioritisation, where no least performance bound exist if the total load from all higher prioritised flows exceeds the egress capacity, $\sum_{j>i}\lambda_j^{in}(t) \geq \mu$, and not that of a single flow.

In order to maximally utilise *statistical multiplexing* current packet switched networks per se do not restrict the traffic inserted by sources. This is left to the intelligence of the traffic sources. Long time, the access bottleneck served as effective ingress limiter. Today, the contractually agreed to and paid for *maximum access bandwidth* may be enforced by the service providers. However, static ingress limiting cannot prevent local overload at individual network nodes, at least not for realistic scenarios with reasonable resource utilization.

To achieve satisfactory transport performance, persistent overload at any node has to be avoided by *ingress control mechanisms*, for example TCP. These mechanism respond to the current network state and thereby assure that in the long term the mean ingress load at no node exceeds its capacity. If all flows are controlled in such a way, the sharing mechanisms presented in this section are effective only temporarily, and the end-to-end performance is in average satisfactory. But, there exist dull transport mechanism (protocols) that do not adjust their insertion rate to the current network state. Also protocols that should adjust their rate may malfunction and flood the network. To cope with the *greedy* flows thereby caused, mechanisms alike those presented in section 4.2 are required to locally handle congestion. This is not the task of sharing mechanisms.

The sharing schemes discussed represent symptomatic approaches, and these were evaluated one-by-one to reveal their strengths and weaknesses. In practice, these schemes may be combined into a *hierarchic queueing stack* (HQS), as it may be provided by carrier grade network nodes.

In [109], a white paper of a renown vendor, we found the following remark: *"The very nature of large IP production networks requires that router vendors offer a combination of these approaches if they are to provide a solution that allows you to accurately arbitrate service-class access to output port bandwidth during periods of congestion. Each vendor's implementation seeks to find the correct balance between performance, accuracy, and simplicity."* Figure 4.40 sketches a basic, exemplary

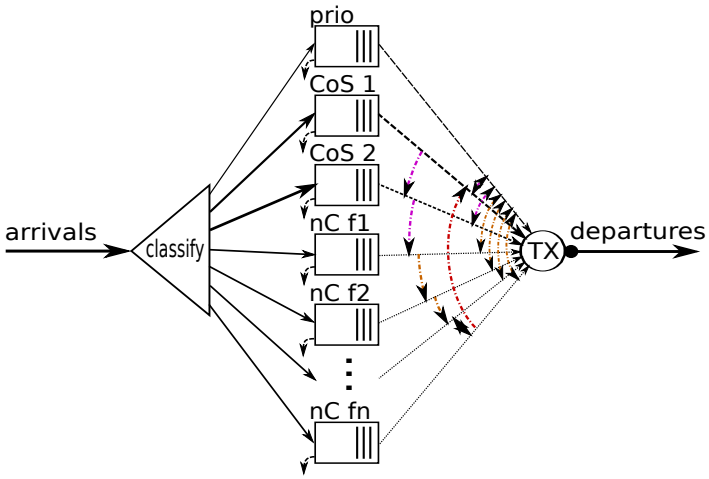


Figure 4.40: A queueing stack integrates basic schemes hierarchically

queueing hierarchy. On top we find *strict priority queueing*, usually only one instance, exclusively used for network internal signalling messages, which for efficiency and performance reasons should anyhow contribute a marginal traffic load only. Below we have some queues that provide *customs based queueing* for accordingly marked packets using the *class of service* (CoS) field in their packet header, followed by a dynamic set of queues used for *weighted fair queueing* of all the remaining flows. While the strict priority queue is served whenever a packet is waiting, and until it becomes idle again, all other queues are served in a *weighted round robin* (WRR) or *weighted random queue* (WRQ) fashion, such that no flow suffers starvation. Thus, if the signalling traffic is well behaved, all flows are served such that the end-to-end communications is never interrupted for unacceptable long periods of time.

Such a stack of schemes enables widely configurable behaviours, too many to evaluate all, and due to the configuring complexity – the configuration of each node influences the optimal configuration of all other nodes – often manufacturer recommendations are implemented in the field. The simplest approach to finding a feasible configuration is assuming that a unique setting for all nodes can be found, and that it can be found iteratively via off-line trail and error based simulation experiments based on 1:1 models of the operational network infrastructures. More sophisticated approaches include *simulated annealing, genetic algorithms, integer linear programming* and methods alike. They all are not applicable on-line due to their unpredictable run-time. A potential alternative, applicable for on-line optimisation, may be *self-organisation* based on *swarm intelligence*. Maybe this can be realised utilising the current research trend toward *software defined networks* [110].

4.2 Congestion management

Congestion results from contention, and identifies a system state where the queue filling considerably exceeds the mean queue filling, causing *unsatisfactory performance*. It occurs when the instantaneous arrival rate $\sum \lambda_i(t)$ exceeds the service rate $\mu(t)$ for a longer than usual period of time, such that a considerable *backlog* of buffered load builds up in the queue, as shown in figure 4.41. To clear a congestion, the arrival rate needs to fall below the service rate for an adequately long time. Literally,

every true queueing system is a congestion based contention management system that implicitly performs statistical multiplexing.

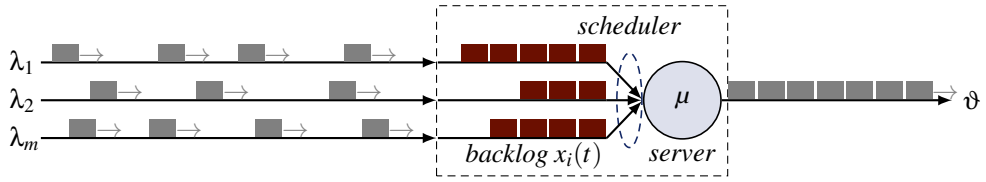


Figure 4.41: Increased backlog of halted load units (packets) during congestion periods

Contention refers to competing resource demands, independent of rates and capacities. The resource sharing schemes outlined in section 4.1 determine the serving in case of contention, when more then one client requests access to the same resource. Thus, more generally, congestion identifies situations where the sharing mechanisms no more provide the intended performance. In the worst case, when due to congestion load becomes relaunched while still waiting in the queue, a *cataclysmic break down*[1] results. Generally, network resources have to be prepared to face congestion every now and than, on the data- or control-plane, if they are not ruled by an almighty central instance that controls every single resource request. Robustness against load peaks is no option, it is a necessary requirement to be fulfilled locally, system by system. Finite queueing systems are per se stable, here congestion management commonly intends to minimise the time required to clear the backlog and to differentiate the negative effects among different flows in a way that protects sensitive flows and appreciates *friendly flows*[2].

In this section we discuss mechanisms that cope with peak loads, irrespective of being temporary (*congestion*) or persistent (*overload*), and implicitly assume that the systems studied are finite. Therefore, we analyse the mean behaviour in overload, and notice that this also constitutes the average behaviour during temporary congestion periods.

First we evaluate queue-filling thresholds in case of shared queues, and how these support differentiated flow serving. Next, we replace the strict blocking above thresholds by increasing drop-probabilities, which introduces *random early detection* (RED), and with weights related thresholds and drop-probabilities *weighted random early detection* (WRED). These local mechanisms interact with the transport control mechanisms implemented at the traffic sources. However, a solid discussion of different transport control protocol (TCP) variants is out of the scope here. Still, basic knowledge of the operation principles of *ingress control* schemes is a prerequisite to understand the interaction with congestion management schemes. Together, smart mechanisms offer considerably more functionality than they can individually. Therefore, a brief introduction to different ingress control mechanisms is included next to the congestion detection mechanisms including a brief statement on how these could as well be implemented distributed toward a scalable and autonomous network control plane.

4.2.1 Queue filling based thresholds

There are two possible approaches. First, thresholds on the individual queue filling per flow $x_i^{\max}$, second, thresholds on the total queue filling $x_{\max}^i$. The first equals individual queues, if the size of the shared queue is sufficiently large, $\sum x_i^{\max} \leq s$, such that all *queue filling thresholds* $x_i^{\max}$ can be satisfied at any time. Else the state transition diagram corner opposing the idle state is cut away, in addition to the individual bounds that limit the state transition diagram dimensions. In that case we get an intermix of shared and individual queues. This difference in queue organisation was already discussed in section 4.1.3. To get the $x_i^{\max}$ based threshold model we set different queue sizes but equal weights. For the mixed system we do not expect very different results. Here we are interested

[1] Looped process that recursively worsens the situation leading to an unresolvable deadlock.
[2] Friendly flows reduce their rate by themselves in case of congestion.

in thresholds on the total queue filling, $x^i_{\max}$. To clearly separate these from the others we refer to them as *queue admission thresholds* in order to implicitly express their function.

If we assume no other differentiation mechanism, the admission threshold controlled system can be modelled by the one-dimensional state transition diagram shown in figure 4.42. This is possible

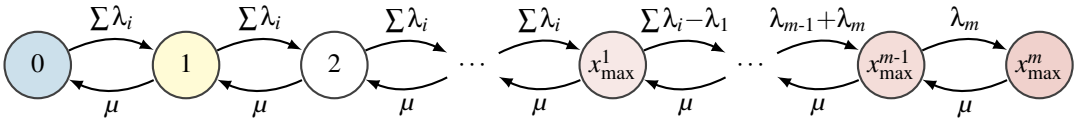


Figure 4.42: Threshold based queue admission control – state transition diagram

because the queue admission threshold controlled blocking does not depend on the mixture of packets present in the shared queue, only on their sum. Composing the Q-matrix representing this system according to figure 4.42 and solving $Q\,p = 0$ with $\sum p_k = 1$, we get the state probabilities p_k.

For each flow, all states above and including the threshold state $x^i_{\max}$ contribute to the blocking probability $P_{b,i}$, from which we can directly calculate the throughputs ϑ_i.

$$P_{b,i} = \sum_{k \geq x^i_{\max}} p_k \qquad \Rightarrow \qquad \vartheta_i = (1 - P_{b,i})\,\lambda_i \tag{4.14}$$

Due to not differentiating the load mixtures that contribute to a state, we cannot calculate the individual queue fillings $\mathrm{E}[X_i]$ straight away, the state transition diagram only provides the total queue filling $\mathrm{E}[X] = \sum k\,p_k$. To achieve the individual queue fillings and all metrics related thereon, we need to reverse the calculation and start with the waiting times $\mathrm{E}[T_{w,i}]$, which we get from the state flow diagram depicted in figure 4.43. The states represent the system at the arrival instant,

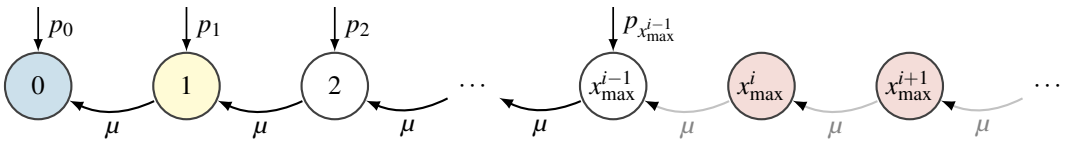


Figure 4.43: Threshold based queue admission control – state flow diagram

prior entry, and thus is the threshold state $x^i_{\max}$ the first blocking state at which an arrival from flow i becomes rejected. Assuming FIFO queueing, a test customer that entered the queue only moves toward the termination state 0, where no others remain in front, and itself is in service. The time that elapses between the entry and reaching state 0 is the random waiting time $T_{w,i}$.

To calculate the mean waiting time $\mathrm{E}[T_{w,i}]$ via the state flow diagram we need to consider that only the arrivals that enter the system contribute, including the arrivals to the idle state, which contribute zero waiting time. Thus, we need to re-normalise the entry probabilities by $p^i_k = p_k / \sum_{j<x^i_{max}} p_j$ before we can use them to proportionately sum the different mean waiting time components $t^k_w = k\,t_h = \frac{k}{\mu}$ that result for entering the queue in state k.

$$\mathrm{E}[T_{w,i}] = \sum_{k<x^i_{max}} p^i_k\,t^k_w = \sum_{k<x^i_{max}} \frac{p_k}{\sum_{j<x^i_{max}} p_j} \cdot \frac{k}{\mu} = \frac{1}{\mu} \cdot \frac{\sum_{k<x^i_{max}} k\,p_k}{\sum_{k<x^i_{max}} p_k} \tag{4.15}$$

The flow times $\mathrm{E}[T_{f,i}]$ than result by adding one holding time $t_h = \frac{1}{\mu}$, and therefrom, using Little's law $N = \lambda T$ with the already calculated throughputs ϑ_i instead of the arrival rates λ_i, as usual for systems with blocking, we finally get the mean system fillings $\mathrm{E}[X_i]$.

$$\mathrm{E}[T_{f,i}] = \mathrm{E}[T_{w,i}] + \frac{1}{\mu} \qquad \Rightarrow \qquad \mathrm{E}[X_i] = \vartheta_i\,\mathrm{E}[T_{f,i}] \tag{4.16}$$

The right side of equation 4.14, and therefore also the right side of equation 4.20, is only correct for Poisson arrivals, where the blocking probability at arrival instances equals the blocking probability at any time $P_{b,i}$, as expressed by the PASTA convention, such that $\lambda_i - \vartheta_i = \delta_i = \lambda_i P_{b,i}$. For other arrival distributions we need to replace each state by an according group of states that reflect the *phase*-type or MAP approximation of the arrival process. In case of multiple flows with different arrival distributions each, this may become excessively complex, though possible. In case of different service rates per flow, μ_i, this approach cannot be used. To cover the influence of the different holding times, we need to consider the composition of the queue filling, which demands the full multi-dimensional state transition diagram as it already has been developed to analyse the different sharing strategies presented in section 4.1.

However, if we know the individual throughputs ϑ_i and system fillings $\mathrm{E}[X_i]$, for example from measurements or by solving the multi-dimensional state transition diagram that considers every possible population mix, we can directly calculate the corresponding mean flow times $\mathrm{E}[T_{f,i}]$ and other metrics in the usual order.

$$\mathrm{E}[T_{f,i}] = \frac{\mathrm{E}[X_i]}{\vartheta_i} \quad \rightarrow \quad \mathrm{E}[T_{w,i}] = \mathrm{E}[T_{f,i}] - \frac{1}{\mu} \quad \rightarrow \quad \mathrm{E}[Q_i] = \vartheta_i \, \mathrm{E}[T_{w,i}]$$

Note that for multi-flow systems we do not get $\mathrm{E}[Q_i] = \mathrm{E}[X_i] - 1$ because the server is occupied by customers from different flows at different times. However, in average the server is occupied aliquot to the throughputs ϑ_i, such that we may calculate $\mathrm{E}[Q_i] = \mathrm{E}[X_i] - \frac{\vartheta_i}{\sum \vartheta_j}$. Inserting this leads to $\mu = \sum \vartheta_i$, which is only true when the system is never empty. Thus, this we can use to get the system's mean idle time $\mathrm{E}[T_{idle}] = 1/(\mu - \sum \vartheta_i)$, which obviously corresponds to the probability of the idle state $p_0 = 1 - \varrho$. Latter is independent of the mix of currently present flows, and as ϱ can be calculated for any μ_i, this also holds for flow specific holding times.

Reconsidering the state transition diagram shown in figure 4.42 we recognise that the total arrival rate to the states above any threshold state is reduced by the arrival rates of the blocked flows. This reduces the probabilities of these states, and due to $\sum p_i = 1$ the states below become more probable than with no thresholds $x^i_{\max} = s$. And in contrast to individual queues opens in congestion every departure a place for an arrival of a privileged flow, irrespective of the flow the departure belongs to. In consequence, the queue gets more and more populated by privileged load. Therefore, small differences in the queue admission thresholds $x^i_{\max}$ cause considerably different blocking probabilities per flow and a considerable reduction of the number of loads from less privileged flows waiting in the queue, as shown in figure 4.44.

Comparing the results with those we get for WRQ+ and individual queues (section 4.1.3, figure 4.25) we recognise increased blocking differences and consequential throughput divergences, at less and also reversed flow time divergences. Therefore, the *queue admission thresholds* are utile if we intend to achieve throughput divergence at comparably similar sojourn times (flow and waiting times representing system and queue sojourn times, respectively). In particular, the sojourn times here are upper bounded by the threshold levels, in that respect privileging flows with lower thresholds, whereas with WRQ+ the sojourn times of less weighted flows can exceed the queue size related benchmarks due to the more frequent serving of higher weighted flows.

Note, to correctly simulate this shared queue system it was necessary to define a queue selection mechanism that strictly enforces the *first come first serve* (FCFS) policy in order to correctly model the operation of the shared FIFO queue. Why was this necessary, because to measure the individual queue filling and sojourn times we choose to use virtually independent queues, i.e., X-, W-records per flow. Any other selection policy causes diverging results if it does not perfectly reflect FCFS.

The new selection option is specified as `SQ` and replaces round robin `RR` as the default policy for shared queues. However, for a single queue `RR` remains the least performance wasting selection algorithm when no selection is actually performed.

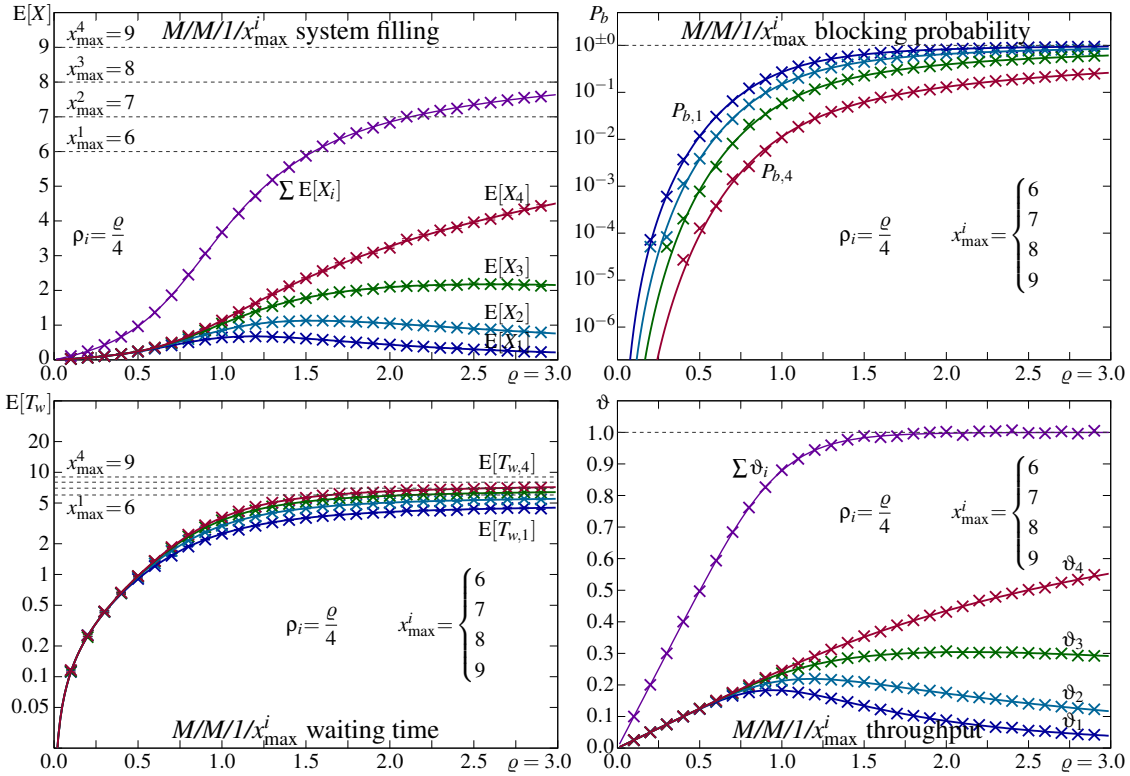


Figure 4.44: Threshold controlled queue admission (simulation: $\times$, calculation: solid lines), shared queue with size $s=9$, Poisson arrivals, equal negative exponentially distributed holding times T_h, equal load shares $\rho_i=\frac{\varrho}{4}$, and thresholds $x^i_{\max}=\{6,7,8,9\}$ for flows $i=\{1,2,3,4\}$ respectively

SQ selection enforcing strict FCFS ***(across multiple queues)***

```
for i = 1 : n do                          ▷ compose a list of current maximum waiting time per flow 1 : n
    if isempty(W(i){:}) then
        wlist(i) ← −1                     ▷ insert negative value (−1) if no client of flow i is waiting
    else
        wlist(i) ← W(i){:}(1)             ▷ or the maximum from the waiting times record W(i)
    end if                                ▷ first (1) is maximum because later arrivals are appended to the end
end for
nextQueue ← find(wlist = max(wlist))(1)   ▷ index of maximum specifies the queue to serve next
```

To see how the admission threshold based system responds to non-Markovian arrival and service processes we repeat the simulation with the distributions already used to evaluate the different sharing mechanisms in section 4.1.3. Please refer thereto for details on the here used *Lomax* and *Beta* distribution and see section 2.2 for these distribution's specific features. The performance results are shown in figure 4.45. Again, the simulation results of the non-Markovian system are compared with the numeric results of the equivalent Markovian system. The results do not diverge much. Expectedly, the $P_{b,i}$ are higher for bursty arrivals, and thus the mean ϑ_i, $\mathrm{E}[X_i]$ and $\mathrm{E}[T_{w,i}]$ are lower.

The arrival and departure coefficient of variation c_A, c_D for both, $M/M/1/x^i_{\max}$ and $L/\mathscr{B}/1/x^i_{\max}$, are shown in figure 4.46. For the Markovian system we again recognise a slight increase of the departure coefficient of variation once the system is overloaded. Interestingly, we find $c_{D,3}\sim 1$ and not $c_{D,4}\sim 1$. The reason therefore is unknown, potentially it is only a result of the specific example settings. With $L(2,1)$ arrivals the monitored $c_{X,i}$ are not stable, as can be expected for infinite variance. However, in overload we recognise a tendency toward $c_{D,i}<c_{A,i}$.

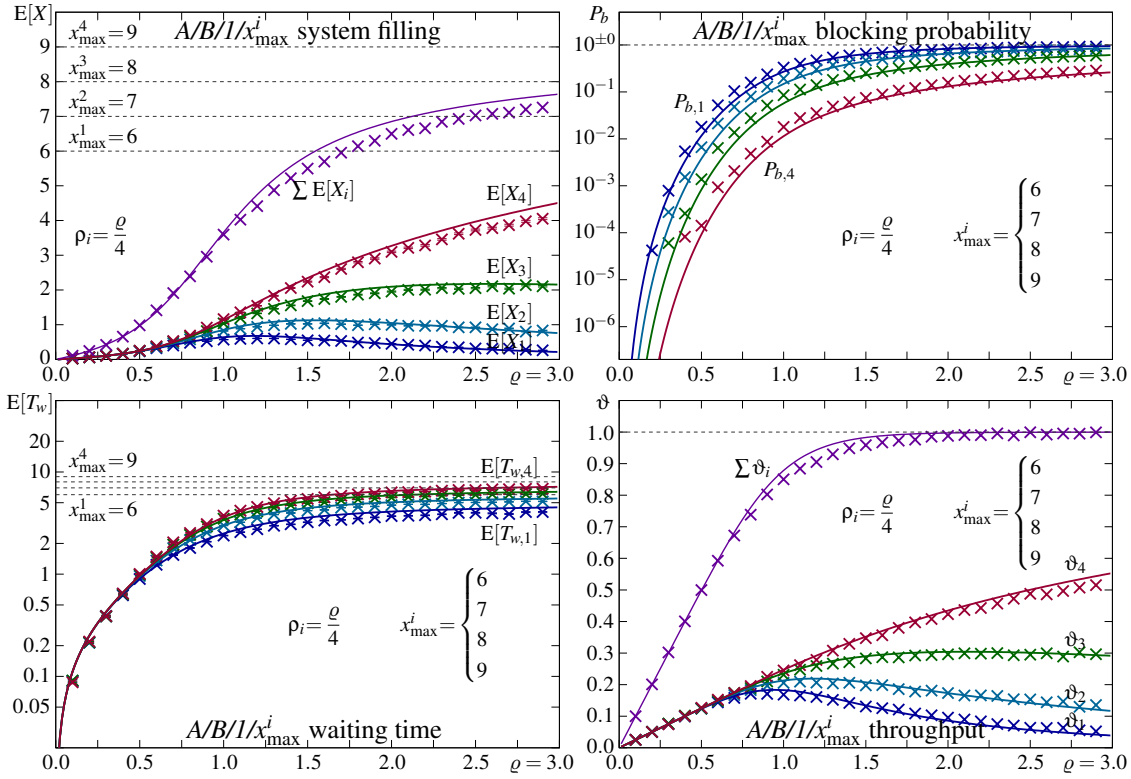


Figure 4.45: $L/\mathscr{B}/1/x^i_{\max}$ simulation ($\times$) compared to $M/M/1/x^i_{\max}$ calculation (solid lines), shared queue with size $s=9$, Lomax (Pareto-II) arrivals $L(2,1)$, Beta $\mathscr{B}(\frac{1}{3},\frac{2}{3},\frac{1}{2},2)$ distributed holding times T_h, equal load shares $\rho_i=\frac{\varrho}{4}$, and thresholds $x^i_{\max}=\{6,7,8,9\}$ for flows $i=\{1,2,3,4\}$

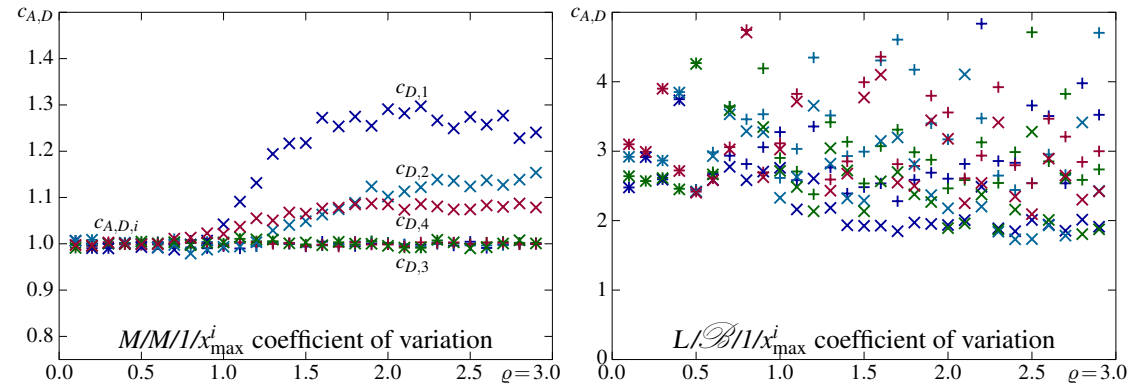


Figure 4.46: Arrival ($+$) and departure ($\times$) coefficient of variation of the $M/M/1/x^i_{\max}$ and $L/\mathscr{B}/1/x^i_{\max}$ system shown in figure 4.44 and figure 4.45 respectively

Combined flow weighting with queue admission thresholds

In section 4.1.3 we recognised the near to optimal behaviour of *weighted random queue* scheduling (WRQ+) and that shared queues are more efficient than individual queues. However, WRQ+ with a shared queue yields only sojourn time differences. The blocking probabilities and the resultant throughputs are identical for all flows, independent of the assigned weights (figure 4.23).

Above we have seen that for a shared queue a differentiation in blocking probability $P_{b,i}$ and throughput ϑ_i can be achieved nicely by *queue admission thresholds* $x^i_{\max}$. If we add these to the WRQ+ scheduler, we get the results shown in figure 4.47. To highlight the features of this combination we choose weight and threshold settings comparable to those used for CBQ in section 4.1.3, figure 4.32. Only the queue thresholds are adjusted to consider the strong effect of the

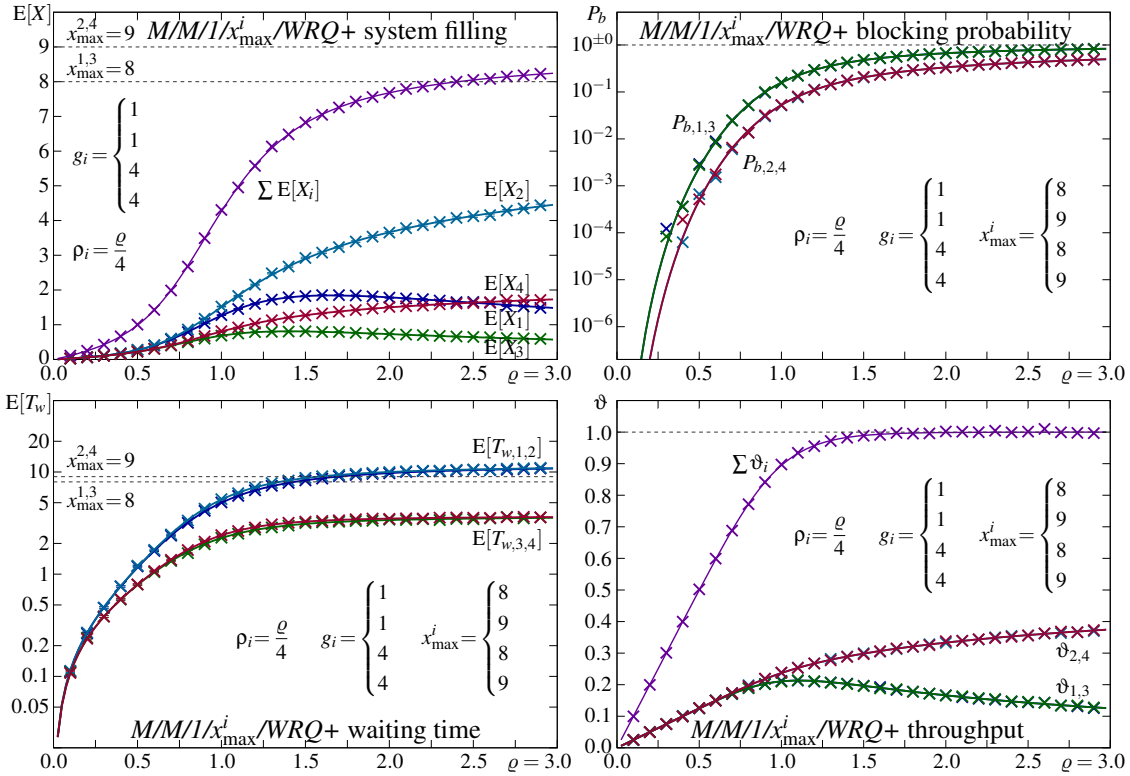


Figure 4.47: WRQ+ with threshold controlled queue admission (simulation: $\times$, calculation: solid lines), shared queue with size $s=9$, Poisson arrivals, equal negative exponential T_h, equal load shares $\rho_i=\frac{\varrho}{4}$, weights $g_i=\{1,1,4,4\}$ and thresholds $x^i_{\max}=\{8,9,8,9\}$ for flows $i=\{1,2,3,4\}$

admission thresholds $x^i_{\max}$ compared to the individual filling thresholds $x_i^{\max}$ used with CBQ.

The mean queue filling $\mathrm{E}[X_i]$ looks a little strange. However, the results for the blocking $P_{b,i}$ and waiting time $\mathrm{E}[T_w]$, gained for the example's settings, show perfect separation of the two differentiation mechanisms. The flows with equal admission thresholds face equal blocking, and thus achieve equal throughput shares, independent of the assigned weights. On the other side, the different weights cause a sojourn time differentiation with equal delaying for equal weights, independent of the admission thresholds assigned. Concluding, this is a scheme that enables quite independent differentiation in blocking and delaying. In addition, it is also efficient in terms memory demand because only a shared queue is required, which is better utilized than individual queues would be.

The example case yielding the performance shown in figure 4.47 assumes equal ingress load shares ρ_i per flow, and evaluates the four primary service types, which may be called *best-effort, low-loss, low-latency*, and lets say *majestic* service. In figure 4.48 we show results for different load splits and different threshold assignments. In the first case, shown on top, the load split follows the recommended convention to always keep a higher privileged traffic volume below that of the next lower prioritised. The overload is than in average caused by the less prioritised flows. This is effectively mitigated by the lower queue admission thresholds $x^i_{\max}$ assigned to less prioritised flows. Higher privileged flows are hardly affected by the overload, meaning their throughput still rises quite linear with increasing individual load and also their flow times are only marginally risen.

If we reverse the load split, such that more load arrives from privileged services than from less privileged, than we get the *worst case*. This is shown by the two graphs in the middle of figure 4.48. For all flows we achieve worse overload performance, being increased blocking (deviation from the initial linear throughput increase) and nearly doubled flow times. To compensate this, we reverse the threshold assignment. This solves the sojourn times issue at the cost of high blocking rates for high weighted flows, as shown in the lower two graphs of figure 4.48. It may seem wrong, but *privileged*

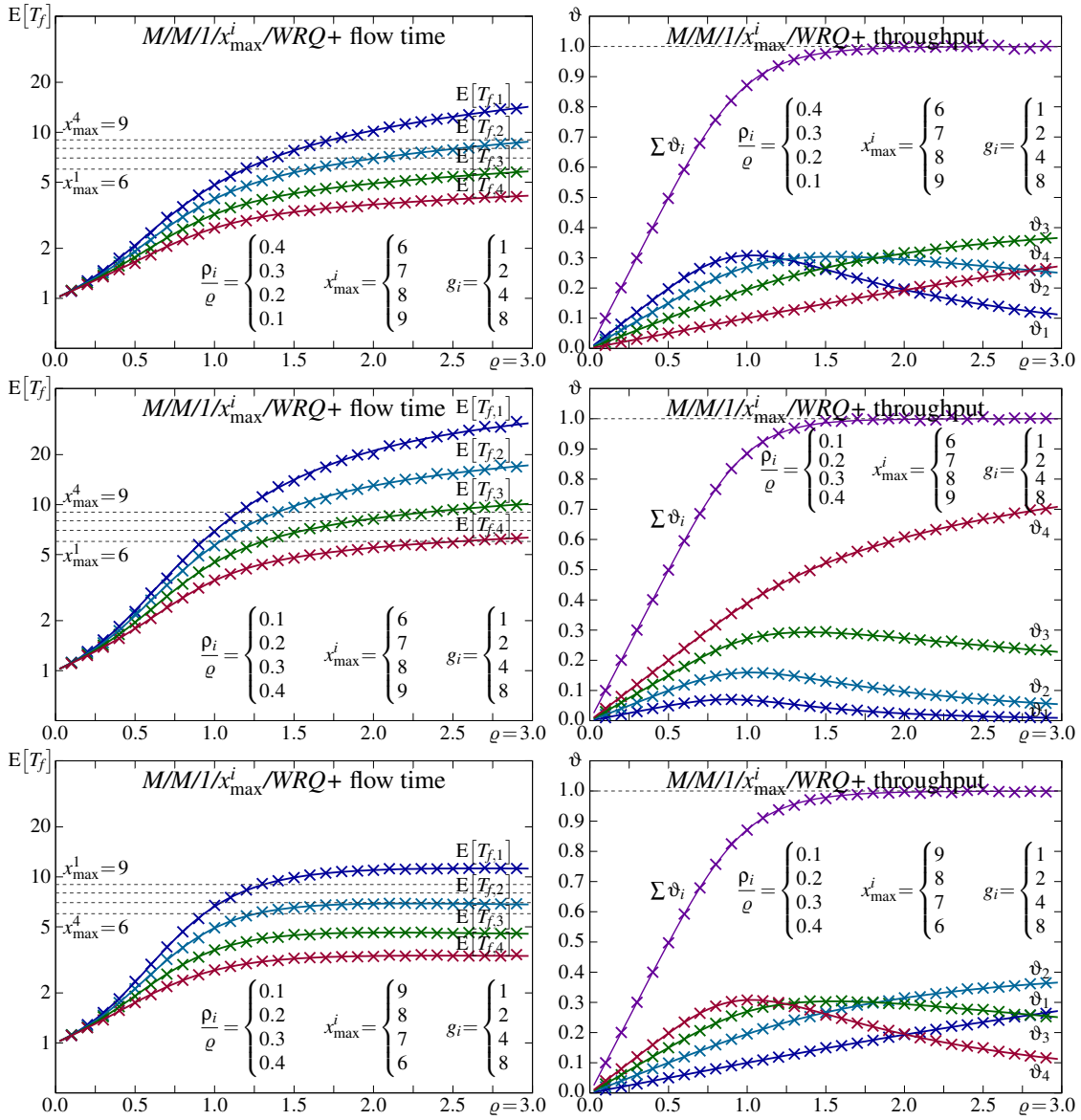


Figure 4.48: *M/M/1/$x^i_{\max}$/WRQ+* flow times $\mathrm{E}[T_{f,i}]$ and throughputs ϑ_i, different load splits $\frac{\rho_i}{\varrho}$ and $x^i_{\max}$ assignments (top: recommended case, middle: worst case, bottom: mitigated worst case)

flows that do not behave have to be confined to their limits in order to keep the system as a whole reliable. For the less privileged flows this confinement is implicitly enforced by the lower weights. Notably, the mitigated worst case yields the smallest maximum delays. However, they are reached earlier, such that below overload the $T_{w,i}$ are higher than in the common recommended case.

To see how one flow changes the situation, we show in figure 4.49 how the performance changes if we increase one flow, $\rho_i = [0..1]$, while keeping the system load constant, $\varrho = 1$, and all other flows contributing an equal share each. At the same time we also introduce the already repeatedly used $Lomax(2,1)$ distributed inter-arrival times T_A and $\mathscr{Beta}(\frac{1}{3},\frac{2}{3},\frac{1}{2},2)$ distributed holding times T_h to the simulation results, in order to assess the sensibility to non-Markovian processes. In figure 4.49 we only show the results for the extremes, changing the least and most privileged flow $i = 1,4$ respectively. The depicted curves may seem unspectacular, but the nearly linear changes show that the combination of *queue admission thresholds* and WRQ+ is neither very sensitive to the actual load split nor to the arrival or service process. Evidently, the blocking is raised by the infinitely variant arrivals and thereby the queue filling and total throughput $\sum \vartheta_i$ reduced. Notably, the waiting time

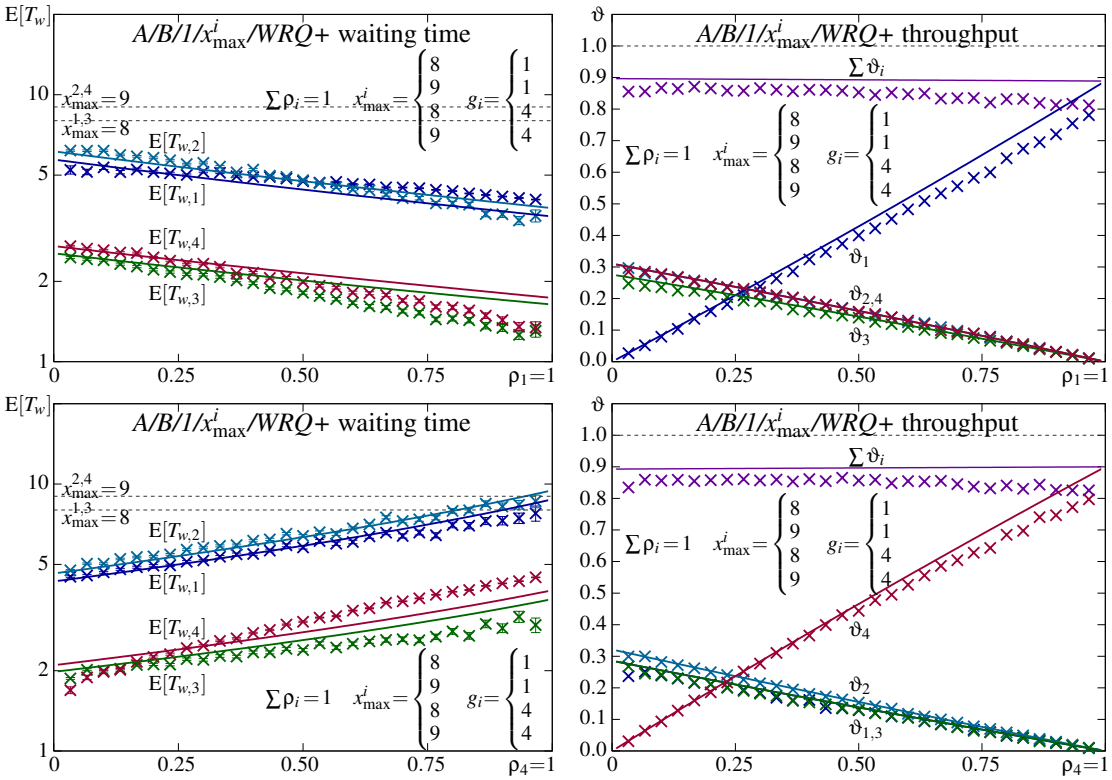


Figure 4.49: $L/\mathscr{B}/1/x^i_{\max}/WRQ+$ simulation ($\times$) compared to $M/M/1/x^i_{\max}/WRQ+$ calculation (lines), Lomax (Pareto-II) arrivals $L(2,1)$, Beta $\mathscr{B}(\frac{1}{3},\frac{2}{3},\frac{1}{2},2)$ distributed T_h, thresholds $x^i_{\max}=\{8,9,8,9\}$, weights $g_i=\{1,1,4,4\}$, one increasing load share, $\rho_i=[0..1]$, remaining load shares $\rho_{j\neq i}=(1-\rho_i)/3$

deviation from $M/M/1/x^i_{\max}/WRQ+$ depends on the load split. The $\mathrm{E}[T_w]$ of higher weighted flows and in particular that of the flow rising from $[0..1]$ diverge most. However, the service differentiation among flows is rather robust and reliable, recommending this combination for class based service differentiation providing *low-loss* and *low-delay* traffic transportation.

The combination of *queue admission thresholds* and *weighted fair queueing* opens a plethora of options. All the viable loads, weights, and threshold assignment variants open a wide space for evaluations. From the depicted selection the core features of the threshold/WRQ+ combination should be expectable. In particular, we recognise in figure 4.48 that the assigned thresholds should inversely match the expected load share, and that assigning high thresholds to highly weighted flows can be counterproductive if such a *majestic* flow occupies the server too much. In that case all flows suffer, and neither *low-latency* nor *low-loss* serving can be granted. This confirms the well known suggestion to *restrict majestically privileged flows to a really tiny share of the entire traffic volume*, or avoid them entirely.

4.2.2 Early congestion detection

Congestion at a core packet router (gateway) affects all flows instantly and in parallel. In combination with ingress control mechanisms, for example TCP, the timely aligned dropping of packets leads to the synchronisation of the protocol specific ingress load *pumping*. In consequence, the load on the bottleneck oscillates and causes a feedback loop that makes the oscillation persistent. The possibly huge variation per flow causes that the aggregate load across all links used by the affected flows varies periodically, which synchronises the flows passing a different bottleneck. In the end, the entire network is bothered by load oscillations (*global synchronisation*). This macroscopic behaviour of TCP and methods to target it are for example presented in [111, 112].

In 1989 E. S. Hashem discussed in [113] a first *random drop* policy and *early random dropping*, targeting the basic synchronisation issue. In 1993 Sally Floyd and Van Jacobson presented an implementation [114], called *random early detection* (RED), which became very popular and is widely available nowadays. A reworked version targeting some flaws of RED never became published, reputedly due to a disliked illustration. A draft of this paper can be found on line [115].

Random early detection (RED)

The name *random early detection* refers to an important feature, being the detection of potential congestion prior becoming critical. In cooperation with sensitive flow-control mechanisms at all traffic sources, RED can effectively avoid serious congestion. However, to perform *congestion notification based ingress management* the traffic flows need to be acknowledged (bidirectional) flows, else the sources never become aware of anything.

The local congestion detection is based on monitoring the local queue filling. To allow short load peaks to pass, RED uses an *exponential weighted moving average*

$$\bar{x}(t^+) = \bar{x}(t^-) + \omega_q \left(x(t) - \bar{x}(t^-)\right) \tag{4.17}$$

adjusted whenever a packet enters the queue, where the weight $\omega_q \leq 1$ determines the time constant of the averaging [114]. If ω_q is chosen too small, the average does not respond sufficient quickly to load changes. If ω_q is chosen too large, short load peaks cannot pass unharmed, causing overreaction. Thus, ω_q needs to be chosen according to the volatility of the traffic and the peak size that may pass without marking. For an efficient implementation a negative power of two is recommended, typically in the area of $\omega_q = 2^{-8}$ and below.

The dropping probability $\delta(\bar{x})$ evolution over the averaged queue filling $\bar{x}(t)$ is shown in figure 4.50. We notice that up to a lower threshold $\bar{x}_{low}$ no dropping is performed. This reflects the

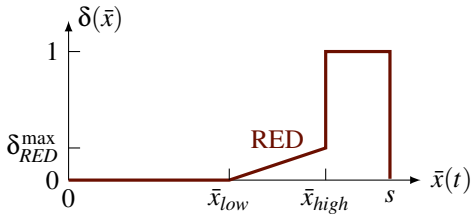


Figure 4.50: Drop probability over averaged queue filling as used with RED

region where no congestion is anticipated. Above this limit the initial dropping probability $\delta(\bar{x})$ slowly increases, until at the upper threshold level $\bar{x}_{high}$ it abruptly increases to 100%. The low dropping probabilities up to $\delta_{RED}^{\max}$, reached at the upper threshold $\bar{x}_{high}$, are sufficient because sensitive sources commonly respond to a packet loss by a drastic (exponential) cut-back of the ingress rate, for example halving it in case of common TCP [RFC793]. That the upper threshold $\bar{x}_{high}$ needs to be well below the available queue size s is necessary because the averaged queue filling $\bar{x}(t)$ does not specify the actual queue size s required to reach this maximum average filling.

The decision to drop a packer or not can be implemented by a decremented counter, which is reset to the inverse of the current dropping probability $\delta(\bar{x})$ whenever a packet has been dropped.

In case of non-responsive or malfunctioning sources the queue will be operated in average at the upper limit $\bar{x}_{high}$. Thus, this needs to be set to a level enough low to grant satisfyingly small delays. The lower threshold than needs to be set to a level sufficiently below that, such that the ingress rate cut-back of the sources takes effect prior the average queue filling reaches the upper bound.

Thus, the exact thresholds and the width of the interval $[\bar{x}_{low} .. \bar{x}_{high}]$, where RED randomly drops packets, depend on the characteristics of the local traffic aggregate, and the macroscopic *round-trip-time* (RTT). Latter specifies the time that passes in between launching a packet and receiving the

counter based dropping decision

```
if c && --c then                          ▷ counter c exists and decremented c > 0
    enqueue the arrived packet
else                                      ▷ counter c = 0 or decremented c = 0
    drop the arrived packet
    c = (int) 1/δ(x̄)                      ▷ reset counter c
end if
```

acknowledgement thereof, and equals the time between a congestion detection at some node and the reduction of the ingress load becoming effective at the according node, as sketched in figure 4.51. In

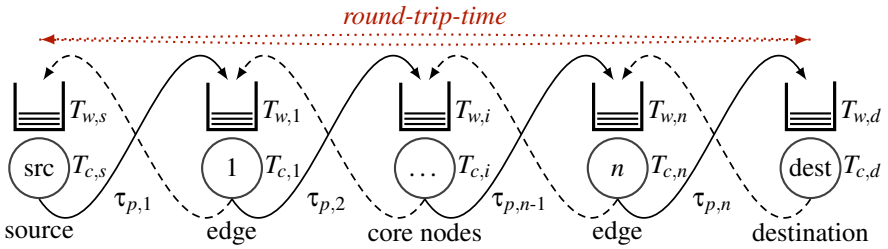


Figure 4.51: Congestion notification and response cycle-time equals the round-trip-time

a *store-and-forward* based packet switching environment the RTT is a *random variable*, composed of a minimum duration defined by the propagation delays $\tau_{p,i}$, a typically dominating component reflecting the various synchronisation and computing delays $T_{c,i}$, and a rather negative exponentially distributed component introduced by the waiting times $T_{w,i}$ spent in buffers (queues). In [114] it is recommended to assume RTT = 100 ms and the thereto related RED thresholds are proposed. In the subsequent literature many approaches intended to solve the unfairness problem of RED related to differing RTT per flow were proposed, for examples see [116–118].

Note, if packet *marking* is supported by the transport technology, packets need not be dropped, they get marked only. Either using a dedicated *congestion notification flag*, or the *excess flag*. Both are used at downstream nodes to keep the node in a comfortable operation region: above a defined queue filling threshold marked packets are simply dropped. The message upon the potential congestion somewhere along the path reaches the destination in any case. Either the destination does not receive all packets sent, missing sequence numbers in case packets actually became dropped, or via the flag. Depending thereon the destination informs the source either by not acknowledging all sent packets, or by notifying it upon the reception of flagged packets.

Anyhow, flows that are not controlled by the source, for example *user datagram protocol* (UDP) flows [RFC768], do not cut-back their ingress rate. In the worst case the applications using such greedy transportation re-send the lost packets, causing the opposite of the intended ingress reduction. Many nowadays popular services predestined to use UDP may not be designed to cut-back the ingress rate on demand (e.g., audio and video services).

Discriminatory loss distribution among flows

To evaluate here the complex interaction of RED and different transport protocols exceeds the scope, but without, an analysis of RED would be incomplete. Instead, we evaluate a simple *random early drop* policy not being designed to a special kind of transport control. This scheme *distributes the packet losses* of overfilled queues differently among flows, with the intention to assure the demanded QoS per flow. It is *based on the current queue filling* $x(t)$ and does not let pass ugly traffic bursts. Be

reminded that this scheme should not be used with per node traffic aggregates because that can lead to *globally synchronized TCP pumping*.

To evaluate the impact of randomised dropping in respect to the performance of a single node handling different flows in parallel, we use system filling dependent drop probabilities δ_x^i per flow, as for example depicted in figure 4.52. The lower threshold x_{low}^i assigned to a flow defines the lower

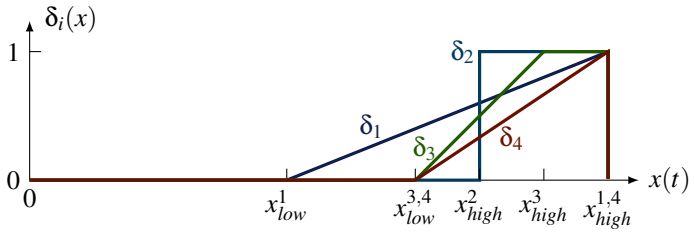


Figure 4.52: Drop probability progression over queue filling

boundary, up to which all arrivals are accepted, non is dropped, and the upper threshold x_{high}^i defines the level from which on no load from the flow is accepted, 100% dropped. If $(x_{high}^i - x_{low}^i) \leq 1$ a hard blocking bound results. The example curves shown in figure 4.52 result for the threshold pairs $(x_{low}^i, x_{high}^i) = \{(4,9)|(7,7)|(6,8)|(6,9)\}$ for the flows $i = \{1,2,3,4\}$ respectively. The length of the shared queue is implicitly given by $s = \max(x_{high}^i)$.

This random early dropping scheme, implementing any dropping probability progression δ_x^i that is based on the current queue filling $x(t)$ only, such that *no memory* is introduced, can be modelled by the state transition diagram depicted in figure 4.53, where $\varepsilon_x^i = 1 - \delta_x^i$ is the fraction at which

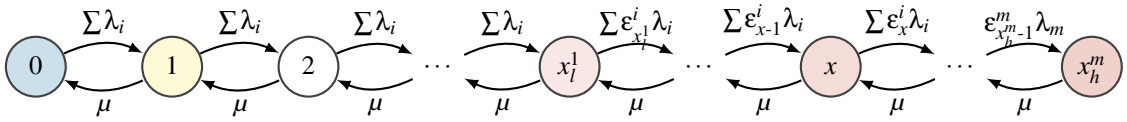


Figure 4.53: Randomised dropping state transition diagram

load from flow i is allowed to enter the shared queue at filling state $x(t)$, for example the linear progressions $\varepsilon_x^i = (x_{high}^i - x)/(x_{high}^i - x_{low}^i)$ shown in figure 4.52. Evidently, this fraction is bound to $0 \leq \varepsilon_x^i \leq 1$. If the value calculated per flow per state falls outside this interval it is to be replaced by the nearest bound, zero or one. Note that for the example definition of ε_x^i no arrivals from flow i are blocked up to and including the lower threshold state x_{low}^i, whereas above and including x_{high}^i all arrivals from flow i are blocked. Accordingly are the blocking probabilities $P_{b,i}$ the sums over the loss probability $\delta_i(x)$ weighted state probabilities p_x.

$$P_{b,i} = \sum_x \delta_i(x)\, p_x \tag{4.18}$$

The mean system filling $\mathrm{E}[X_i]$ and flow times $\mathrm{E}[T_{f,i}]$ need to be calculated via the conditional mean waiting time t_w^x as before, here considering the entry probabilities $\varepsilon_x^i = 1 - \delta_i(x)$.

$$\mathrm{E}[T_{w,i}] = \sum p_x^i\, t_w^x = \sum \frac{\varepsilon_x^i\, p_x}{\sum_j \varepsilon_j^i\, p_j} \cdot \frac{x}{\mu} = \frac{1}{\mu} \cdot \frac{\sum \varepsilon_x^i\, x\, p_x}{\sum \varepsilon_x^i\, p_x} \tag{4.19}$$

$$\mathrm{E}[T_{f,i}] = \mathrm{E}[T_{w,i}] + \frac{1}{\mu} \qquad \Rightarrow \qquad \mathrm{E}[X_i] = \vartheta_i\, \mathrm{E}[T_{f,i}] \tag{4.20}$$

Introducing the entry probabilities ε_x^i simplified the presentation of the equations because we now may sum over all states. Not reachable states contribute zero to the sums because of their $\varepsilon_x^i = 0$.

Using the dropping probabilities sketched in figure 4.52 and equal load per flow we get the results shown in figure 4.54, assuming independent Poisson distributed arrivals per flow and negative exponentially distributed holding times. The different dropping probability progressions were

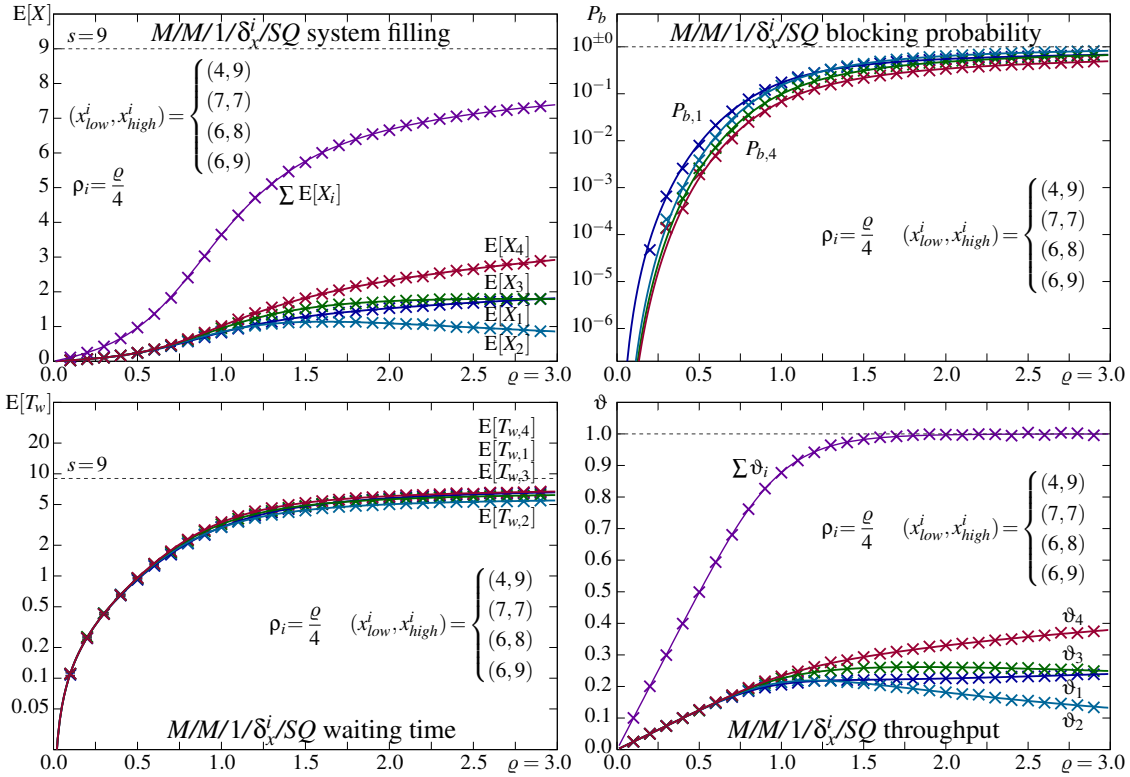


Figure 4.54: Randomised dropping with lower and upper thresholds as shown in figure 4.52 (simulation: $\times$, calculation: solid lines), shared queue with size $s=9$, Poisson arrivals, equal negative exponential T_h, equal load shares $\rho_i=\frac{\varrho}{4}$

chosen to reveal some basic behaviours. First, we notice that the mean waiting time $\mathrm{E}[T_w]$ are quite similar. This is due to the equal service shares (fair sharing) and the rather similar upper threshold. The blocking probabilities P_b are also not very different. At loads $\varrho < 1$ the lower threshold of flow 1 causes increased blocking, whereas in heavy overload the upper threshold dominates the differentiation. Concerning the queue filling and throughput we notice that a less than maximal upper threshold causes decreasing curves. Flows that may use the entire queue never get shares that decrease with increasing system load.

Comparing infinitely variant $L\mathrm{omax}(2,1)$ arrivals and bounded $\mathscr{B}eta(\frac{1}{3},\frac{2}{3},\frac{1}{2},2)$ distributed holding times ($\times$ – simulation) with Poisson arrivals and negative exponentially distributed holding times (*solid lines* – calculation) we get the results shown in figure 4.55, where we assume increasing load shares $\frac{\rho_i}{\varrho}=\{0.1,0.2,0.3,0.4\}$ for the flows $i=\{1,2,3,4\}$ representing the non-recommended case where more load is caused by more privileged services.

The highly variant arrivals cause increased blocking probabilities $P_{b,i}$, which evidently cause reduced mean throughputs ϑ_i but also reduced mean system fillings $\mathrm{E}[X_i]$ that jointly cause slightly reduced mean waiting times $\mathrm{E}[T_{w,i}]$. For the chosen dropping progressions δ^i_x latter differ only marginally. Anyhow, the mean metrics do not divert much from those found with Poisson arrivals and negative exponentially distributed holding times (*lines* – $M/M/1/\delta^i_x$), such that we may conclude that the performance of this dropping system does not depend much on the characteristics of the arrival and service process.

An interesting effect of *random early dropping* is revealed in figure 4.56: the increased departure

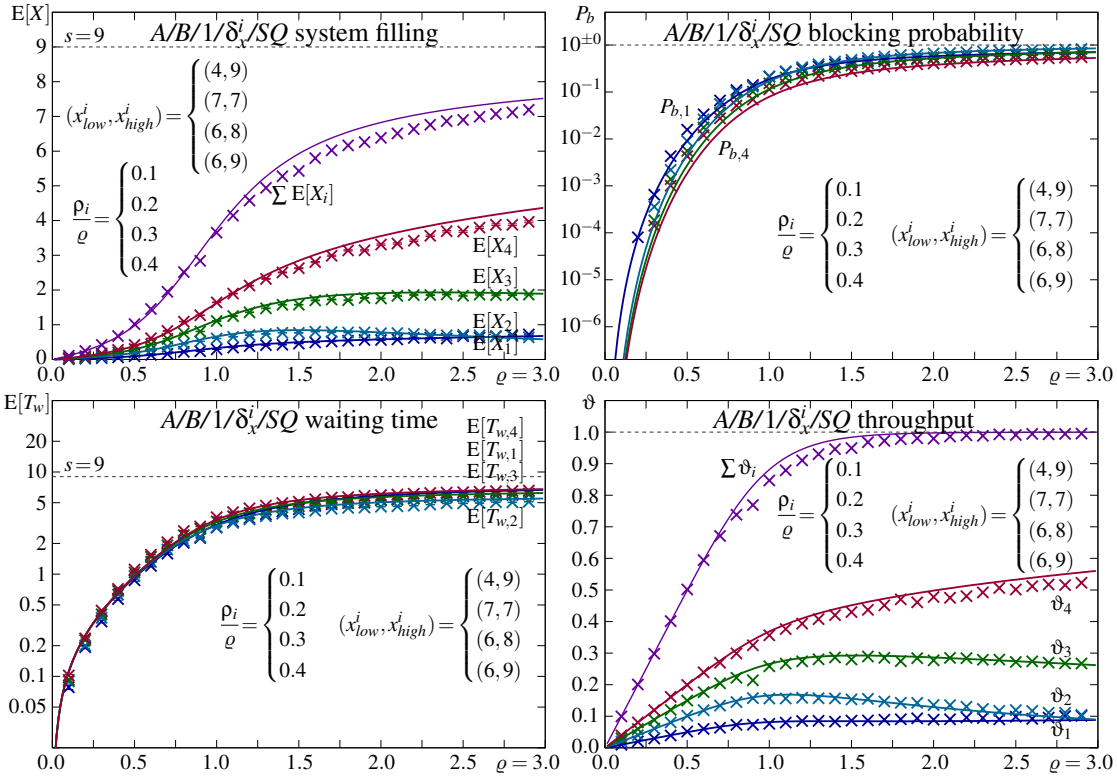


Figure 4.55: Random dropping with the per flow drop progressions δ_x^i shown in figure 4.52, comparing $L/\mathscr{B}/1/\delta_x^i/SQ$ ($\times$) with $M/M/1/\delta_x^i/SQ$ (*lines*), shared FIFO queue ($s=9$), increasing load shares $\frac{\rho_i}{\varrho}=\{0.1,0.2,0.3,0.4\}$ for flows $i=\{1,2,3,4\}$

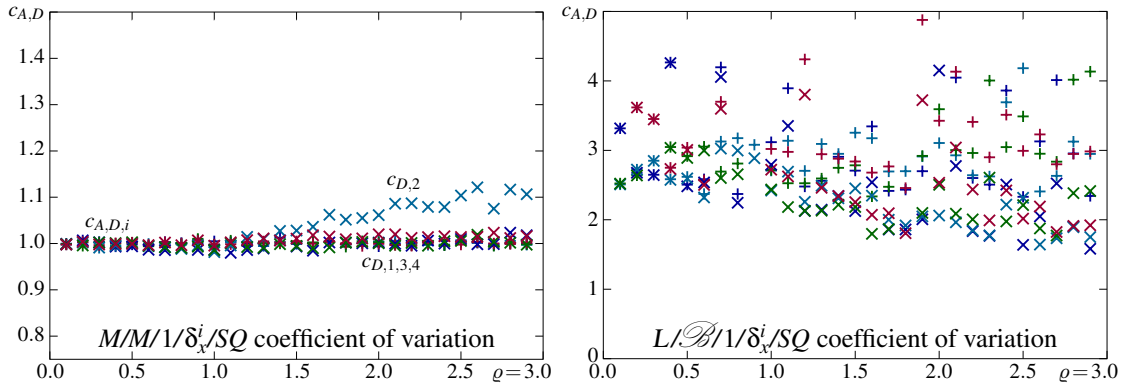


Figure 4.56: Arrival (+) and departure ($\times$) coefficient of variation $c_{A,D}^i$ for $M/M/1/\delta_x^i/SQ$ (left) and $L/\mathscr{B}/1/\delta_x^i/SQ$ (right), as evaluated in figure 4.54 and 4.55 respectively

coefficient of variation visible for hard bounded $M/M/1/x_{\max}^i$ in figure 4.46 vanishes for all the evaluated *dropping progressions* δ_x^i that actually cause random drops prior reaching the upper bound.

Random early dropping combined with weighted sharing – $A/B/1/\delta_x^i/WRQ+$

If we add weighted resource sharing, for example WRQ+ with $g_i=\{1,2,4,8\}$ for flows $i=\{1,2,3,4\}$, we get the results shown in figure 4.57, where we show increasing/decreasing load shares for the flows $i=\{1,2,3,4\}$ on the left/right side respectively. Comparing the case with increasing load shares (left figures) with the calculation results (*lines*) shown in figure 4.55 we recognise that adding WRQ+ primarily introduce a clear differentiation in terms of the mean waiting times $\mathrm{E}[T_{w,i}]$. The blocking probabilities $P_{b,i}$ are determined by the loss progressions δ_i and so are the throughput shares

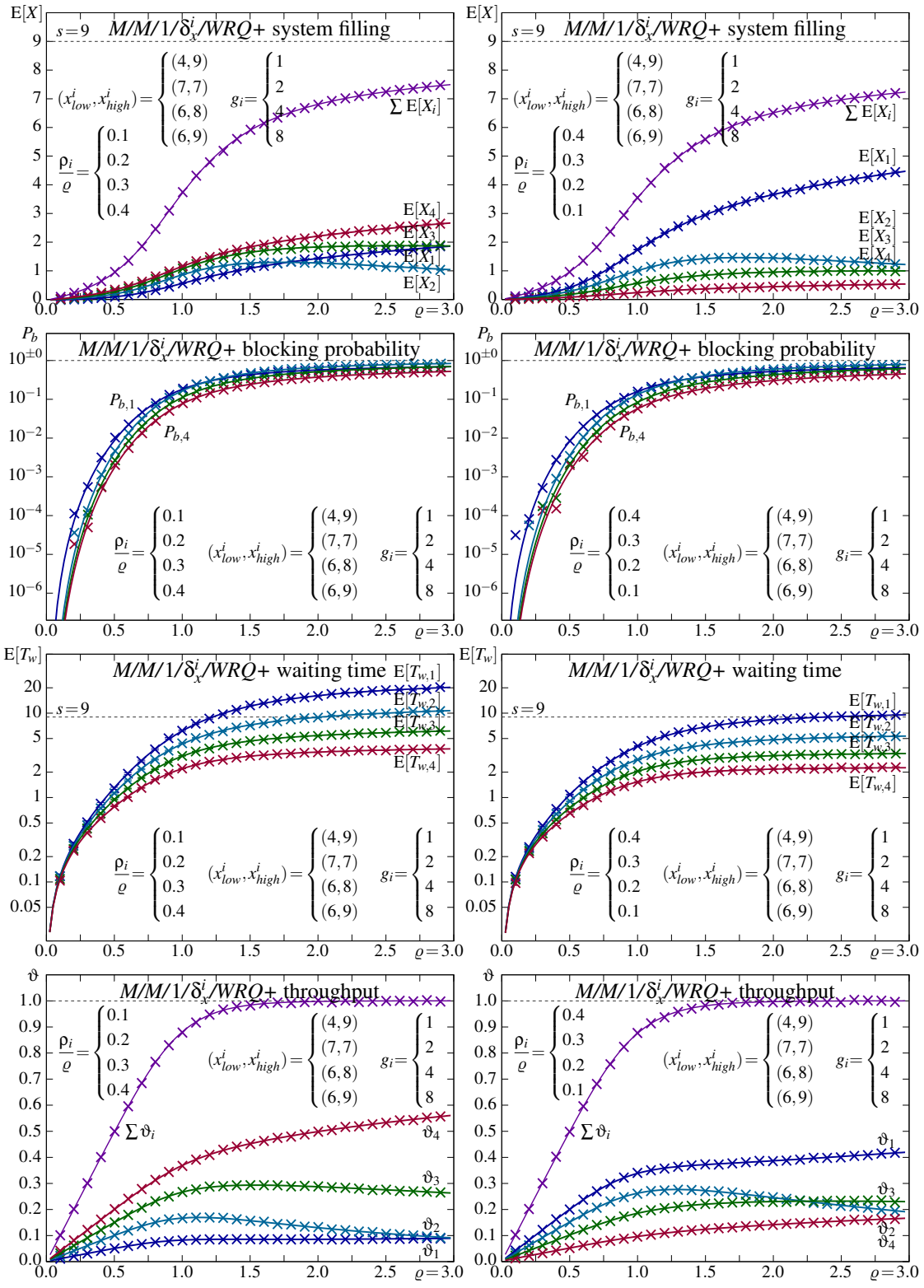


Figure 4.57: $M/M/1/\delta^i_x/WRQ+$ applying the per flow progressions shown in figure 4.52 and sharing weights $g_i=\{1,2,4,8\}$, comparing increasing load shares $\frac{\rho_i}{\varrho}=\{0.1,0.2,0.3,0.4\}$ (left) and decreasing load shares $\frac{\rho_i}{\varrho}=\{0.4,0.3,0.2,0.1\}$ (right) for flows $i=\{1,2,3,4\}$ respectively

ϑ_i. Concerning the different load distributions we again recognise the effect of much load from highly prioritised flows, primarily expressed by the considerably increased mean waiting times $\mathrm{E}[T_{w,i}]$, which we find for all present flows.

Note, to calculate numeric results, the *lines* in figure 4.57, we cannot use the state transition diagram depicted in figure 4.53. We need to return to the multi-dimensional state transition diagram developed in section 4.1.3 (figure 4.21 and 4.22) in order to get the different conditional mean waiting times $t_w^{\vec{x},i}$ per flow i and state $\vec{x}$ in the waiting time calculation $\mathrm{E}[T_{w,i}] = \sum p_{\vec{x}}^i t_w^{\vec{x},i}$. The upper thresholds equal the queue filling bounds used in section 4.2.1, $x_{high}^i \leftrightharpoons x_{\max}^i$. Along the state diagonals beneath the queue size boundary, in between x_{low}^i and x_{high}^i, we have to reduced the arrival rates λ_i by the according dropping probabilities δ_x^i, yielding $\lambda_i(x) = \lambda_i(1-\delta_x^i)$. Also the calculation of the individual blocking probabilities $P_{b,i}$ and mean waiting times $\mathrm{E}[T_{w,i}]$ needs to be accordingly adjusted to correctly consider all contributing system states $\vec{x}$.

If we weight the services with $g_i{=}\{1,1,4,4\}$ for flows $i = \{1,2,3,4\}$ and use δ_1 for flows $1,3$ and δ_3 for flows $2,4$, we get the results shown in figure 4.58, comparing infinitely variant $L\mathrm{omax}(2,1)$

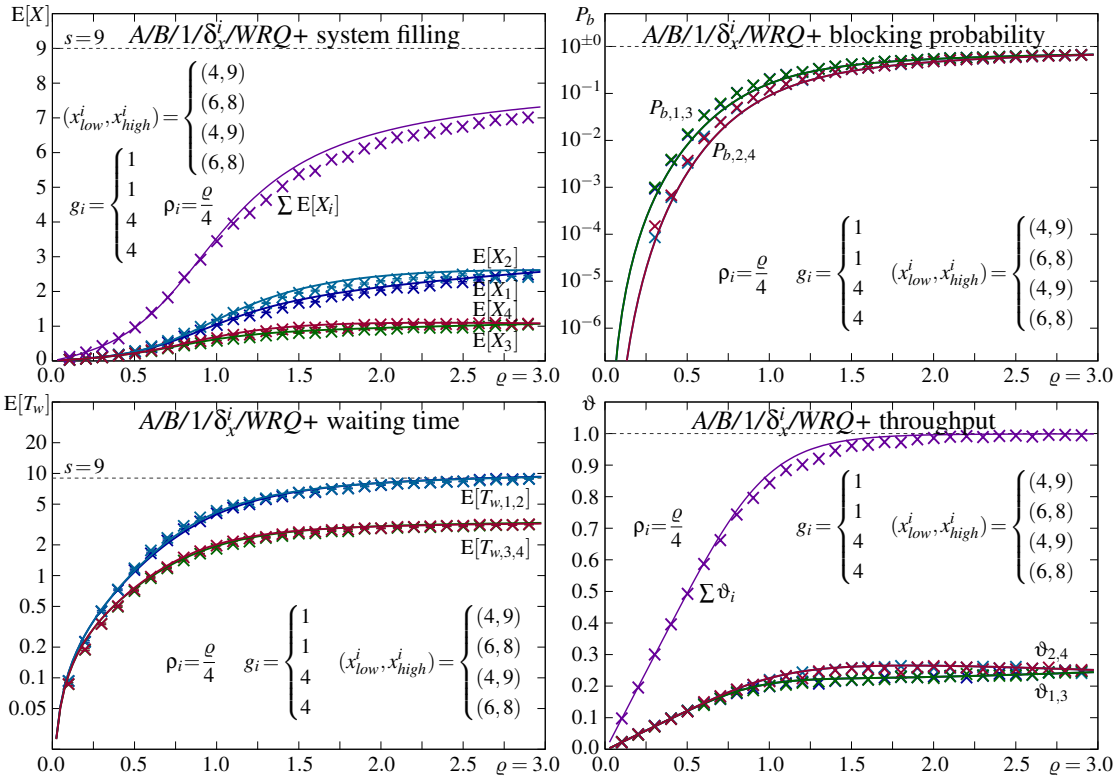


Figure 4.58: $L/\mathscr{B}/1/\delta_x^i/WRQ+$ simulation ($\times$) compared to $M/M/1/\delta_x^i/WRQ+$ calculation (*lines*), with weights $g_i{=}\{1,1,4,4\}$ and dropping progressions $\delta_x^i = \{\delta_1, \delta_3, \delta_1, \delta_3\}$ (figure 4.52) for flows $i = \{1,2,3,4\}$ respectively, and equal load shares $\rho_i{=}\frac{\varrho}{4}$

arrivals and $\mathscr{B}eta(\frac{1}{3}, \frac{2}{3}, \frac{1}{2}, 2)$ distributed holding times ($\times$ – simulation) with Poisson arrivals and negative exponentially distributed holding times (*solid lines* – calculation), assuming equal load shares $\rho_i{=}\frac{\varrho}{4}$ for evaluation clarity.

We recognise a nearly perfect separation of the effects of the different discrimination mechanisms: flows with the same dropping progression δ_x experience the same blocking $P_{b,i}$, whereas flows with the same serving weights experience the same mean flow time $\mathrm{E}[T_{w,i}]$. The divergence caused by the infinitely variant arrivals and bounded service time does not change this separation of the impact caused by the different mechanisms joined in the $A/B/1/\delta_x^i/WRQ+$ queueing system. The results for different load distributions shown in figure 4.57 let us to assume that this clear separation holds for any load distribution.

Concluding remarks on *random early dropping*

The individual definition of x^i_{low} and x^i_{high}, and the shape of the drop probability progression δ^i_x, in between these thresholds, offers in theory infinite options. The upper thresholds work alike the thresholds evaluated in section 4.2.1, whereas the δ^i_x introduce the randomisation and thereby the option to respond to potential congestion before it becomes fatal. If perfectly configured for all flows, the *detection earliness*, meaning the difference between x^i_{low} and the saturated queue, provides the flows' terminals a chance to jointly react in a way that avoids unsatisfactory QoS due to excessively high loss rates and huge latencies from 100% utilised queues.

Depending on the ingress control mechanism different δ^i_x shapes need to be applied. A single shape that serves all control mechanisms contradicts the opposing responses of the different control strategies commonly applied. The placing of the upper and lower thresholds should consider the mean round-trip-time $\mathrm{E}[RTT]$ of the flow this dropping progression is applied on, else the response to a potential congestion may become effective too late to prevent the collapse, or so early that other flows see no need to reduce their rates as well. In figure 4.59 these problems become apparent: if

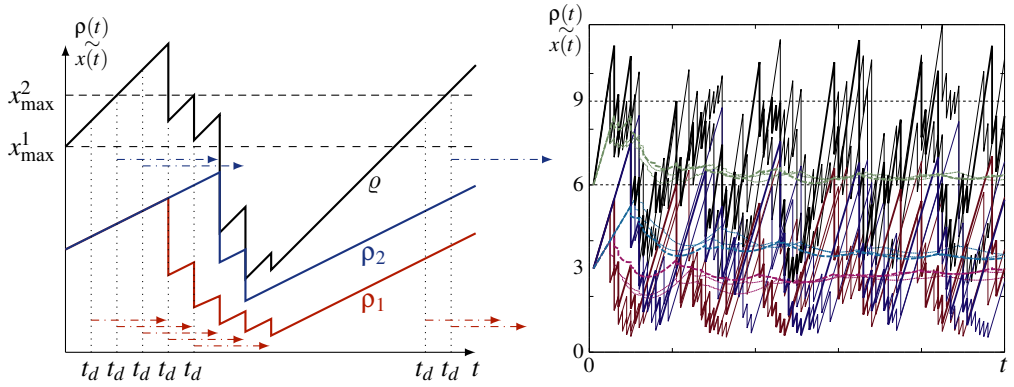


Figure 4.59: Threshold controlled ingress load over time for greedy applications: hard bounds $x^{1,2}_{\max}$ and deterministic $\mathrm{RTT}_{1,2}=\{3|4\}$ (left), and equal with δ_4 randomised bounds and $\mathrm{RTT}_{1,2}=\{2|8\}$ (right), assuming TCP alike transmit window size $t_w(t)=k(t)\,t_h$, and deterministic holding times $t_h=1$

mean RTT_i times the maximum load times the current window size exceeds the difference between lower and upper threshold, $\mathrm{RTT}_i\,\rho_i\,t_w > (x^i_{high} - x^i_{low})$, the buffer filling exceeds the upper threshold prior the rate reduction becomes effective. Due to the time delay introduced by RTT, the rate is reduced also after the current queue filling $x(t) \sim \rho(t)$ dropped below the lower threshold, causing heavy fluctuations. If we observe the control process over longer time and with random dropping the results get messy (right figure). However, the cumulatively calculated mean reveals that flows with shorter RTT get in average a lower share, known as the *TCP/RED unfairness issue*.

Traditionally, core packet routers (gateways) are not aware of the flows the packets they forward belong to. With MPLS this changes. If well configured, the random packet losses introduced by *early random dropping* affects only one flow at a time, such that synchronisation among flows sharing a label switched path (LSP) should not occur. For these also the unfairness issue does not apply because their RTTs are identical. Thus, *early random dropping* applied with per flow thresholds and a local flow based sharing policy, for example WFQ, should not cause *global synchronisation*. In addition, any per flow enforced sharing policy will evidently annul any unfairness issues related to different RTTs of different LSPs.

4.2.3 Ingress control

In section 4.1 we introduced resource sharing mechanisms, and in the subsections above, methods to cope with overload (congestion). We evaluated these systems primarily under heavy overload because that is the situation where these mechanisms massively influence the performance of a queueing

system. We noted, that such overload situations occur temporarily also for mean loads below the system capacity, meaning for $\sum \lambda_i < \mu$, whenever random processes are involved. Actually, sharing and congestion management mechanisms are primarily designed for these temporary occasions only, and not to handle persistent overload. How to avoid persistent overload, is the task of *ingress control* mechanisms, briefly presented in this section to complete the resource sharing topic. These mechanisms operate at client side, above the transport layer, and control the ingress load prior it is actual launched for transportation across the network.

Ingress control relies either on accurate knowledge of the entire network state, as for example assumed with the *reservation based* approach utilising the *request-check-grant* control paradigm, or it relies on *feedback information* provided either by network components, in particular its management or control plane, or received from the remote destination entity receiving a transmitted data flow, implementing the *trial-and-adjust* control paradigm.

Every feedback, irrespective what component provides it, has to travel back to the source across the network, as shown in figure 4.60. The path does not need to be the reversed forward path, and in

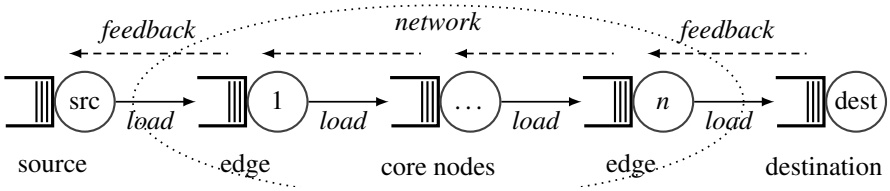


Figure 4.60: Control feedback provided either by network nodes or the remote destination

any case, the passed queues are others because in our model we model the transmit queues, which evidently are not the same for $i \to j$ and $j \to i$. Thus, feedback is in general randomly delayed, and the control algorithm should take care of the thereby introduced uncertainty.

The common transport control protocol (TCP), invented in line with the internet protocol (IP) and briefly introduced in section 4.2.2 in conjunction with the random early detection mechanism (RED), is a sophisticated example of latter. Note that IP without TCP is in theory, assuming infinite queues, not stable. In 1992, A.K.J. Parekh proved that for sessions that are shaped at the edges of the network by token or leaky bucket rate control, weighted fair queueing (WFQ) can provide strong upper-bound, end-to-end delay performance guarantees [109], rendering these ingress control mechanisms particularly important for reliable QoS provisioning.

Commonly, admission control performs authentication and conformity with some *service level agreement* (SLA). Latter commonly includes a maximum insertion rate besides the customer privileges. But how can a maximum insertion rate be specified for packet based load, and what happens if it is exceeded? The ingress rate is the first derivative of the ingress load, which happens to be infinite at packet arrival instances. Actually, common SLAs specify a peak rate only, and that is defined as a mean rate over some short times span, in the area of at least one maximum packet length. Based thereon the *leaky* and *token bucket* ingress limiters can be designed, as shown shortly.

More recently admission control is envisaged to be used as well to adjust the characteristics of data flows in order to provide at any time a service that smartly serves the application: low quality, hopefully still acceptable, when the network is in trouble, and good quality, but not more than necessary, else. This principle is known as *application aware* service provisioning, *smart networking* and alike, and states a fundamental idea of the *next generation network* (NGN) stratum approach, which separates the network operation form the service provisioning [2].

Reservation based ingress control

Temporary congestion can be avoided only if the ingress flows are strictly limited such that the local sum over all ingress flows never exceeds the capacity of the systems passed, such that

$$\sum_i \zeta_{ij}\,\lambda_{i,\max} < \mu_j \quad \forall_j \tag{4.21}$$

where i is the flow index, j the resource index, and ζ_{ij} a binary indicator: $\zeta_{ij} = 1$ if flow i uses resource j, else $\zeta_{ij} = 0$. If the load distribution over time and per flow is far from deterministic, meaning $\mathrm{E}[\lambda_i(t)] \ll \lambda_{i,\max}$, such a restriction causes unsatisfactory underutilisation of resources. In addition, to assure the condition, the limits of all flows need to be adjusted whenever a new flow is routed, or flows need to be rejected if no route can be found for which the condition is fulfilled. Latter is exactly the operation scheme of traditional *circuit switching*, where deterministic line-rates are provisioned per channel, irrespective of the transported load's distribution.

If temporary congestion is accepted in the favour of *statistical multiplexing*, equation 4.21 can be relaxed to mean rates $\lambda_i = \mathrm{E}[\lambda_i(t)]$. This exemplifies *virtual circuit switching*, as it is commonly used with multi-protocol label switching (MPLS), where resources are not strictly assigned, only their mean availability is granted.

A principal problem of any *request – grant* based approach is the a priory required knowledge upon flow statistics and the signalling of the demand prior any traffic can be transmitted. For heavily varying traffic loads, for example on-off modulated traffic streams, this is not very suitable: reserving a capacity share sufficient to cover the on-rate causes unsatisfactory poor resource utilization during off-periods, whereas reserving the mean rate will not yield the required transmission performance during on-periods.

However, the reservation based approach is widely used to route label switched paths (LSPs) in an off-line fashion. The capacity of LSPs is yielded by an *utility maximisation* algorithm, such that the resources are best utilised but not overloaded.

$$\max |\vec{\mu}| \quad \textit{constraint to} \tag{4.22}$$

$$\sum_i \zeta_{ij}\,\mu_i \leq c_j \quad \forall_j \tag{4.23}$$

$$\mu_i \geq d_i \quad \forall_i \tag{4.24}$$

The equations 4.22, 4.23, and 4.24 express it mathematically as a *linear programming* problem, where for clarity we replace the resource's service rates μ_j by the capacities c_j, and the load of LSPs λ_i by the provided mean transmission rates μ_i, which represent the maximised values.

The problem with this approach are the binary ζ_{ij}. They represent the LSP routing and if that is part of the optimisation problem, it becomes a *mixed integer linear programming* (MILP) problem, which is not as simple solvable. Note, *multi-constraint routing* is in general NP-hard [119]. Please see the rich literature on both, MILP and constraints based routing, if interested in this topic. However, if we exclude the routing from the optimisation, for example by recklessly routing all LSPs along shortest paths irrespective of the capacities required/available or assuming the routing to be performed on demand in between re-optimisation instances, the problem can be solved in polynomial time. Latter represents a feasible compromise that anyhow results if we apply the *never break a working connection* paradigm, widely being the default strategy of lower layer control and here commonly implied.

The optimisation defined by equations 4.22 maximises the total capacity of all LSPs, being the length of the LSP-*rates vector* $\vec{\mu}$. Alternative optimisation targets include but are not limited to: $\max(\vec{\mu} - \vec{d})$, $\max(\vec{\mu} - \vec{d})^2$, $\max(\min \mu_i)$, $\max(\min(\mu_i - d_i))$, where each has its particular pros and cons. Primarily is the optimisation constraint by 4.23 to the capacities available per resource, the *resource vector* $\vec{c}$. The other constraint specified by 4.24, which assures minimal rates per LSP,

the *demand vector* $\vec{d}$, is actually optional. The resultant capacities per LSP, the $\vec{\mu}$ components, are evidently not the capacities actually required to transport the current data flows. Instead, they state the long-term mean rates at which these LSPs are capable to transport flows.

Each LSP itself becomes a *queueing system* where the service rate is now a random function defined by the competition levels of the underlying resources. Thus, this approach yields a *virtual topology* of label switched routers (LSRs) connected by the LSPs i that can handle each a certain load, the optimal μ_i, without the risk to persistently overload any resource along the LSP. If per LSP ingress limiting is implemented, the virtual queueing network operates in stable conditions and can be analysed alike any other network of queueing systems (section 4.3).

However, to correctly consider the impact of prioritisation or discriminatory resource sharing, individual LSPs need to routed for every possible QoS demand and source-destination pair. This is neither scalable nor efficient, if we consider that rarely all service classes are required between any two nodes, particularly not for all times. To cope with this, LSPs may be used to transport different service classes jointly, and each LSR performs discriminatory resource sharing in order to establish different service qualities along LSPs. We can imagine how predictable this approach is, considering that we may not be able to predict the load mixture, neither in terms of the QoS demands nor concerning the aggregate flow statistics. Anyhow, the design of virtual topologies is not covered here, neither henceforth. Please refer to the rich literature thereon if interested in this approach to network optimisation.

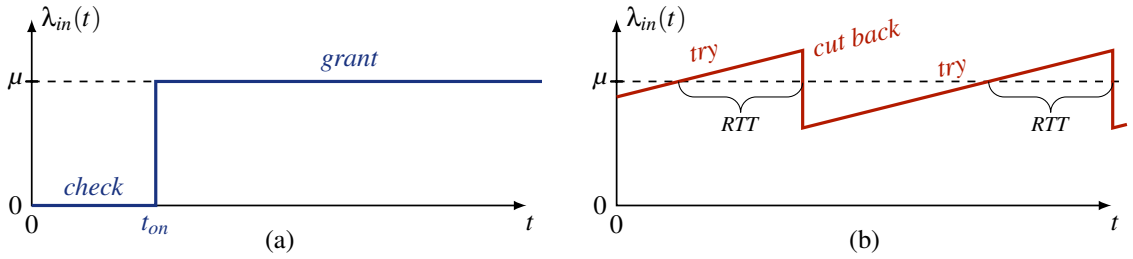


Figure 4.61: Exemplary ingress rate $\lambda_{in}(t)$ of greedy applications managed by opposing control paradigms: a) reservation based *request-check-grant*, and b) feedback based *trial-and-adjust*

In figure 4.61 the principal drawback of the two control paradigms is depicted: reservation based ingress control causes a delayed transmission start whereas feedback based ingress control cannot grant constant transmission rates.

In practice, *request-check-grant* based resource assignment is natively applied for *circuit switching*, and in different forms to establish more or less *circuit like* channels across per se connection-less network technologies, for example using Ethernet's Stream Reservation Protocol [120]. The *effective bandwidth* approach has been widely used, in particular for the *asynchronous transfer mode* (ATM) technology [121] because it is very utile to cover data flows that we can sufficiently well assess, not only in terms of their mean traffic volume but also the statistics of the data units transported, for example fitting a *phase-type model* as presented in section 2.1.4. However, for varying traffic aggregates, as they commonly occur on inter-domain connections and LSPs carrying different flows at different times, this is not possible. In these cases we can only approximate the flows characteristics very roughly by a worst case assumption, meaning calculating an effective bandwidth for the worst flow characteristics that may occur. Note, any effective bandwidth calculation implicitly presumes time invariant traffic characteristics, which rarely reflects temporary flow aggregates well.

Polling based ingress control

For packet switched networks in general, the *request-check-grant* approach is rather awkward: it contradicts the principal *launch-and-forget* approach and thus, limits the scalability drastically. A viable alternative are *polling systems*, where the resource is temporarily assigned to serve a flow exclusively. This approach is very similar to resource sharing as presented in section 4.1. Here the focus is more on the switch-over policy and that we do not know prior polling a queue if load is waiting or not. The resource serves a queue (load source) either one packet at a time, a restricted amount of packets or time per visit, or until the ingress queue of the served flow becomes idle. Latter is called *exhaustive* and is fallacious if $\lambda_i > \mu$ is not a priori prevented. In general, polling results in *on–off* modulated serving per source, as shown in figure 4.62. The resource is effectively shared by a self-managed random time division multiplexing strategy. We note that the total ingress rate is the

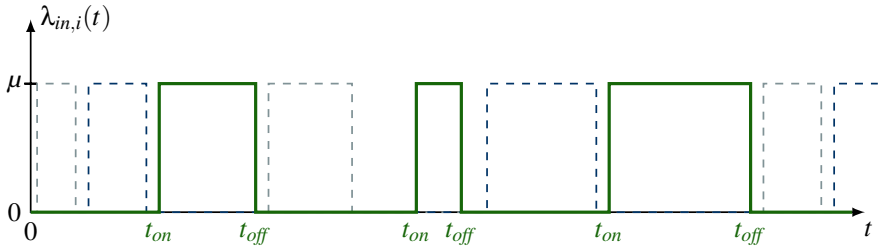


Figure 4.62: Exemplary ingress rate $\lambda_{i,in}(t)$ of polled data streams ($\sum_i \lambda_i < \mu$)

sum of interleaved curves (dashed), which themselves are similar to each other, separated by short gaps caused by the switch-over delay where no source is served. In case an empty queue is polled the gap increases by an additional switch over time. Longer, randomly distributed idle intervals result when all ingress queues become empty. In consequence is the gross ingress process a randomly interrupted deterministic process in term of the line-rate, and an interrupted Poisson process (IPP) in terms of the packet-rate, if the packet lengths L are negative exponentially distributed.

This polling approach is very feasible for confined environments based on a single shared medium, for example an access line, a radio cell, or the switching core inside a router. Multiple nodes constituting a communication network linked by individual independent transmission capacities per link, are a different beast to cover. If interested in shared medium networks, please refer to the rich literature on polling systems, for example see [60] and the literature referred to.

Note, an LSP used by multiple flows also comprises a shared medium, although a virtual one with random serving capacity distributed with $\mathcal{F}(R)$, where R is the *residual LSP capacity*. For deterministic load unit sizes ℓ, being constant packet-/frame-/cell-lengths, this yields random holding times distributed by $\mathcal{F}(T_h) = \ell\ \mathcal{F}(R)$. For randomly sized load units distributed with $\mathcal{F}(L)$, the holding time distribution for the queueing model representing an LSP is the convolution of the unit-size distribution with the service rate distribution, $\mathcal{F}(T_h) = \mathcal{F}(L) * \mathcal{F}(R)$. If $\mathcal{F}(T_h)$ can be specified, we can actually use any of the models presented in chapter 3 to analyse how well an LSP can transport a flow or the models presented in section 4.1 for a group of flows sharing the LSP. However, also the residual mean capacity $\mathrm{E}[R]$ of the underlying resources may changes over time, background load dependently as well as due to re-routing of LSPs in order to adjust the virtual topology to changed demands. Thus, evaluating LSPs as shared medium will commonly yield results that are either only snapshots valid for a particular network state, not easily derivable into generally valid results, or, rough means that are observable only over very long times, and thus, rather irrelevant for flows with a much shorter life-time.

Leaky bucket based rate limiting

Flow control based on the leaky bucket principle implements a strict upper bound on the ingress rate [122]. This is given by the chosen egress rate μ_b of the bucket. While the arrival rate $\lambda(t)$ exceeds μ_b, arriving clients become buffered in the bucket. When the bucket is full, arriving clients become discarded. Thus, the departure rate never exceeds the bucket rate, such that

$$\vartheta(t) \leq \mu_b \; \forall_t. \tag{4.25}$$

Evidently, if $\lambda(t) > \mu_b \; \forall_t$ the departure rate equals the bucket rate, $\vartheta(t) = \mu_b \; \forall_t$, because the bucket never becomes empty. At practical loads with mean $\lambda < \mu_b$ clients are buffered temporarily only. Short load peaks exceeding μ_b are replaced by intervals during which the output rate equals the bucket rate. In between load peaks, once the bucket emptied, the output rate equals the arrival rate, as we know it from common queueing systems. However, here the term rate commonly refers to a constant *bit-rate r*, not a mean packet-rate μ, and the buffer size q to *load volume* stated in [bit/byte] or [seconds], and not the number of clients s (*loads count*). Latter becomes irrelevant if the queue can be assumed infinite, and can else be approximated by $s \approx \frac{q}{\mathrm{E}[L]}$, where L is the random packet size. The former translates as usual into a random holding time T_h distributed alike the packet lengths L, scaled by the constant line-rate: $T_h = \frac{L}{r}$. Assuming Poisson arrivals and negative exponentially distributed holding times, the leaky bucket limiter can be modelled by a simple *M/M/1/s* queueing system with service rate μ_b and queue size s, as presented in section 3.2.

To evaluate it a little more generally, let us assume batch arrivals, where a random number of packets arrive simultaneously at randomly distributed arrival instances, as it for example occurs with file transfers and web-page access. The state transition diagram representing this case is that of *GI/M/1/s*, shown in figure 4.63. The splitting of the arrival transitions is determined by

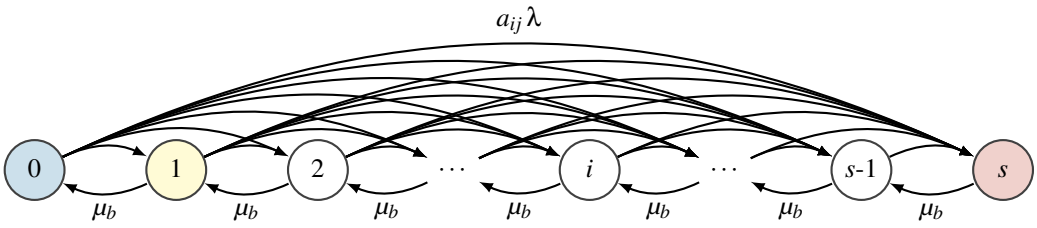


Figure 4.63: $M^M/M/1/s$ queueing model used to evaluate the *leaky bucket* ingress limiter

the probabilities a_n for n arrivals at an arrival instance, which here are Poisson distributed with $\mathrm{P}[n] = (\alpha^n/n!)\, e^{-\alpha}$ (equation 2.1 for $\lambda_{\text{batch}} = \alpha$ and $\tau_{\text{batch}} = 1$). In case the batch size n exceeds the available space, the bucket is filled to its limit s and only the excess part of the batch is blocked. Thus, the arrival transition weights a_{ij} and blocking probabilities per state β_i result as

$$a_{ij} = \begin{cases} \dfrac{\alpha^{(j-i)}}{(j-i)!}\, e^{-\alpha} & i \leq j < s \\ \displaystyle\sum_{n=s-i}^{\infty} \frac{\alpha^n}{n!}\, e^{-\alpha} = 1 - \sum_{k=i}^{s-1} a_{ik} & j = s \end{cases} \tag{4.26}$$

$$\beta_i = \sum_{n=s-i+1}^{\infty} \frac{\alpha^n}{n!}\, e^{-\alpha} = 1 - \sum_{n=0}^{s-i} \frac{\alpha^n}{n!}\, e^{-\alpha}. \tag{4.27}$$

Note that zero batch size may occur ($a_{ii} = e^{-\alpha} > 0$) and thus, the sum over all arrival transition probabilities exiting a state to a state above differs by that amount from one, $\sum_{j>i} a_{ij} = 1 - a_{ii} = 1 - e^{-\alpha}$. However, using these equations we can fill the transition matrix Q as usual, yielding the upper triangular Q-matrix typical for general arrival processes. Solving the linear equation system

defined by $Q\,p = 0$ we get the state probabilities π_i and therefrom the system properties (lines) shown in figure 4.64 for different bucket sizes $s = \{3|9|15\}$. As usual, the blocking probability P_b decreases

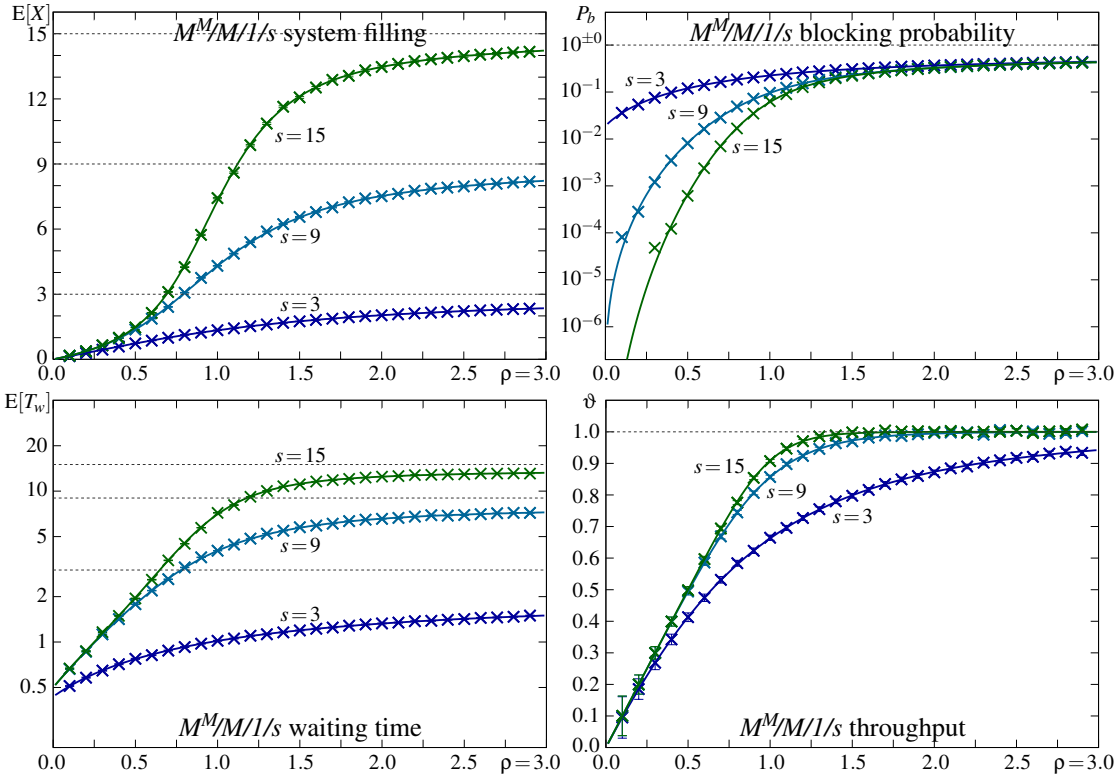


Figure 4.64: $M^M/M/1/s$ leaky bucket ingress limier performance, μ_b=1

with increased bucket size, whereas the mean waiting time $\mathrm{E}[T_w]$ increases. However, due to the batch arrivals they are finite (>0) also for $\rho \to 0$. Changing the bucket rate μ_b causes a rescaling of the x-axis, shown in figure 4.64 for μ_b=1, only. For the example arrival process we choose a mean batch size α=1, such that the arrival rate of batches λ determines the mean system load $\rho = \frac{\lambda}{\mu_b}$. For a different mean batch size α, the batch arrival rate λ needs to be accordingly adjusted to get the same load, $\rho = \frac{\alpha\lambda}{\mu_b}$. The perfect match with simulation results ($\times$) proves that the $M^M/M/1/s$ queueing system actually models the evaluated leaky bucket ingress rate limiter, and we may conclude that this holds as well for any arrival and service processes, if we use $GI/G/1/s$ to model it.

Token bucket based rate limiting

Flow control based on the *token bucket* principle implements a bound on the mean rate $\mathrm{E}[\vartheta(t)] \le \lambda_b$, and not on the peak departure rate. This bound is given by the token rate λ_b at which tokens arrive to a dedicated tokens-bucket of size s_b, as sketched in figure 4.65. Every time some traffic unit

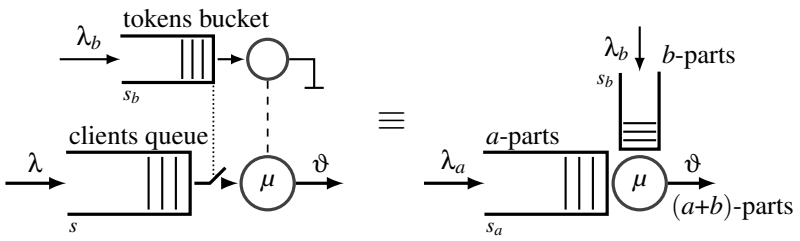


Figure 4.65: Token bucket ingress rate controller architecture (principle)

(client) is served, an according number of tokens is taken from that bucket (dashed line). When no

token is available, or not sufficiently many to serve the next waiting load unit, the server pauses until enough tokens are available (dotted line). This resembles a joining process along an assembly line, where two individually delivered parts $\{a,b\}$ become a single joint part $(a{+}b)$ forwarded to the next assembly station (the sketch on the right in 4.65). Thus, to analyse this system we need a model that comprising two arrival processes and two individual queues: the common load arrivals and buffering queue, and the name giving tokens-bucket, a queue that buffers the tokens. Note, in practice this system is usually not implemented with two queues. As usual, there exist manifold effective realisations that perform equal in respect to the control measures but with less implementation effort.

In practice the tokens are commonly configured to arrive deterministically at constant rate r_b, granting the transmission of a particular load volume [bits/bytes] each. As before, we can replace these volume based tokens by tokens that arrive distributed according to the load volume distribution per load unit, the holding time distribution, and grant the transmission of an entire load unit of random size L per token. Thereby, we remain in the *per packet regime* commonly assumed with packet queueing models. In case L is negative exponentially distributed, the arrival rate of these virtual tokens becomes a Poisson process with $\lambda_b = \frac{r_b}{\mathrm{E}[L]}$. In average, this yields the same limited mean throughput $\mathrm{E}[\vartheta]$.

As the process utilises two queues we need a two dimensional state transition diagram to model token bucket ingress control. This is sketched in figure 4.66, for convenience considering per packet tokens and assuming that tokens depart as the load unit that used it departs (*departure after service* or *late departure regime*). Arriving loads cause transitions to the right with rate λ, arriving tokens

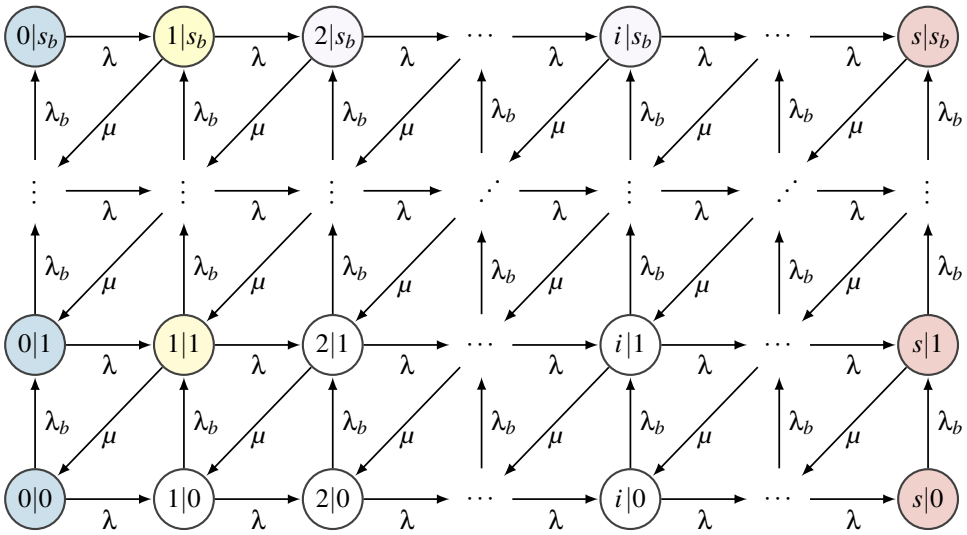


Figure 4.66: *Token bucket* ingress control state transition diagram (*late departure regime*)

cause upward transitions with rate λ_b, and load departures cause diagonal down left transitions with the service rate μ because they free one unit from both queues, the served load unit and the token consumed. If no tokens are available no serving occurs. Therefore, state $(1|0)$ is a waiting state, whereas states $(1|1{:}s_b)$ do not contribute to the mean waiting time, alike all states with zero clients in the system (left edge). On the other side, all states with $i{=}s$ contribute to the blocking probability (right edge). The boundary on tokens (upper edge) has no direct performance relation, it only limits the number of load units that can be served in between two token arrivals, $\tau_b = \frac{1}{\lambda_b}$, which in consequence yields the burst duration and size bounds state shortly in equation 4.28.

Using the state transition diagram shown in figure 4.66 we can fill the Q-matrix and solve $Qp = 0$ to get all state probabilities p_i and from these the system properties (lines) shown in figure 4.67 and 4.68 for different token arrival rates $\frac{\lambda_b}{\mu} = \{0.2|0.5|1|2\}$, bucket size $s_b{=}6$, and a load queue of size $s{=}9$. In figure 4.67 both, tokens and loads arrive negative exponentially distributed because

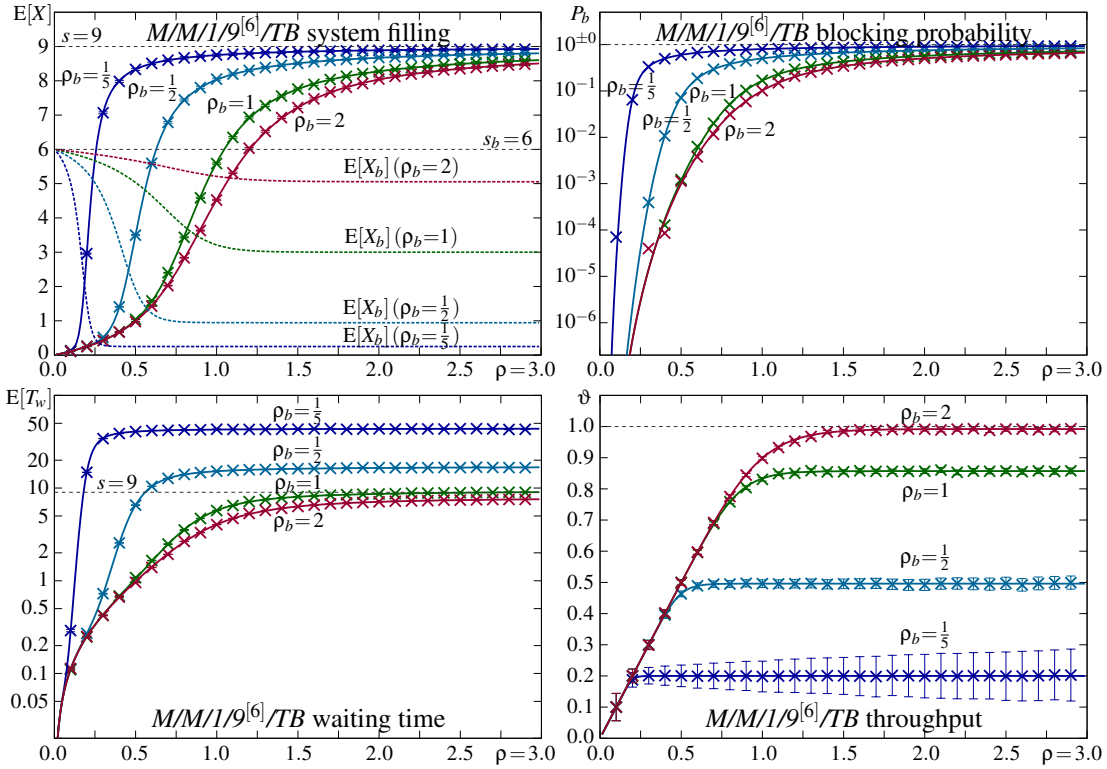


Figure 4.67: $M/M/1/9^{[6]}$ token bucket ingress controller, negative exponentially distributed inter-arrival and holding times, token arrival rates $\rho_b = \frac{\lambda_b}{\mu} = \{0.2|0.5|1|2\}$, for $s=9$, $s_b=6$, and $\mu=1$

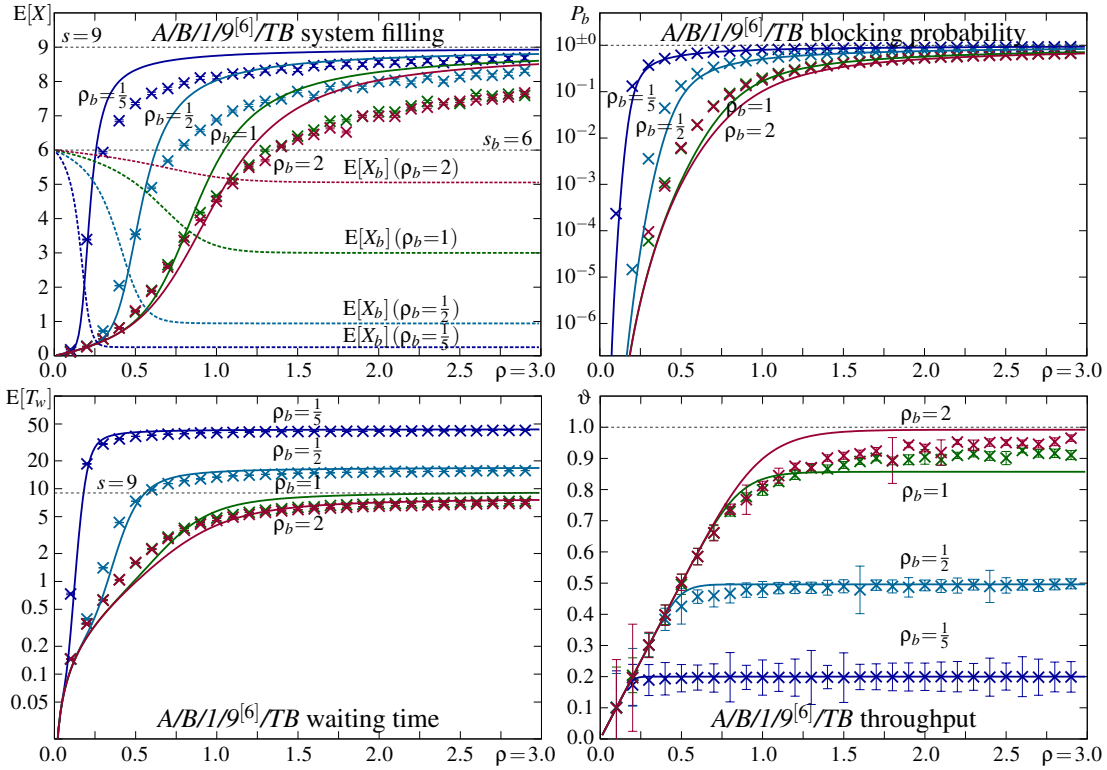


Figure 4.68: $L/\mathscr{B}/1/9^{[6]}$ token bucket ingress controller, $Lomax(2,1)$ distributed load arrivals, $\mathscr{B}eta(\frac{1}{3},\frac{2}{3},\frac{1}{2},2)$ distributed T_h and token arrivals (simulation – $\times$), compared with $M/M/1/9^{[6]}$ results (numeric – lines), token arrival rates $\rho_b = \frac{\lambda_b}{\mu} = \{0.2|0.5|1|2\}$, for $s=9$, $s_b=6$, and $\mu=1$

for the model we assume that the tokens arrive distributed alike the holding times, which here are assumed to be negative exponentially distributed. The perfect match approves the model and vice versa the simulation, although not the assumption that deterministic load volume tokens can be modelled by packet tokens distributed alike the holding time. However, we recognise that a token rate to service rate relation $\rho_b = \frac{\lambda_b}{\mu} > 1$ does not much but still visibly improve the performance. In particular the reduced achievable throughput for $\lambda_b = \mu$ is conspicuous. It presumably results from the stochastic dis-alignment of load and token arrivals. For mean rates considerably smaller, $\lambda_b \ll \mu$, the ingress controller effectively reaches the target rate, where $\vartheta = \lambda_b$. The price are considerably increased mean waiting times $\mathrm{E}[T_w]$, which may exceed the buffer size to service rate boundary, $\frac{s}{\mu}$.

In figure 4.68 we compare simulation results for infinitely variant $Lomax(2,1)$ distributed arrivals and bimodal $\mathscr{B}eta(\frac{1}{3},\frac{2}{3},\frac{1}{2},2)$ distributed holding times and token arrivals, assuming $F(A_b) = F(T_h)$, with the numeric results from figure 4.67. The results show that the high variability of arrivals degrades the performance considerably, basically as expected. Interesting is, that the performance gain for $\lambda_b > \mu$ is reduced. This is presumably caused by the smoother holding times and thus smoother token arrivals.

If a token bucket ingress controller is implemented within a customer system providing sufficient buffering capacity (system memory), for example a personal computer or an embedded processor providing several gigabyte of random access memory, enough for a queue length 10^6-times the mean holding time and more, than an upper bound on the loads queue is practically not required. In case the thereby caused potentially very high ingress delay obsoletes a load unit, the application can simply withdraw it from the ingress queue because it is located within the same device, the *traffic source* as seen from the network. Withdrawals may be included in the model, see for example [123–126] on queueing of *impatient/reneging customers*. Here we do not consider load withdrawal and apply the *matrix geometric method* (MGM) outlined in section 3.1.6 to solve the *infinite token bucket controller*. According to the state transition diagram shown in figure 4.66, dropping the right boundary edge, we get the sub-matrices

$$B_1 = \begin{bmatrix} -\lambda-\lambda_b & \lambda_b & 0 & \cdots & 0 & 0 \\ 0 & -\lambda-\lambda_b & \lambda_b & \ddots & 0 & 0 \\ 0 & 0 & -\lambda-\lambda_b & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & -\lambda-\lambda_b & \lambda_b \\ 0 & 0 & 0 & \cdots & 0 & -\lambda \end{bmatrix} \qquad B_2 = A_2 = \begin{bmatrix} \lambda & 0 & 0 & \cdots & 0 & 0 \\ 0 & \lambda & 0 & \ddots & 0 & 0 \\ 0 & 0 & \lambda & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & \lambda & 0 \\ 0 & 0 & 0 & \cdots & 0 & \lambda \end{bmatrix}$$

$$B_0 = A_0 = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ \mu & 0 & 0 & \ddots & 0 & 0 \\ 0 & \mu & 0 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & \cdots & \mu & 0 \end{bmatrix} \qquad A_1 = \begin{bmatrix} -\mu-\lambda-\lambda_b & \lambda_b & 0 & \cdots & 0 & 0 \\ 0 & -\mu-\lambda-\lambda_b & \lambda_b & \ddots & 0 & 0 \\ 0 & 0 & -\mu-\lambda-\lambda_b & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & -\mu-\lambda-\lambda_b & \lambda_b \\ 0 & 0 & 0 & \cdots & 0 & -\mu-\lambda \end{bmatrix}$$

required to apply the MGM equations as outlined in section 3.1.6: in particular, the iterative calculation of the rate-matrix R, equation 3.59, the calculation of the state-probability vectors per level π_i, equation 3.60, and the mean system filling $\mathrm{E}[X]$ calculation, equation 3.62. Using the usual calculation chain $\mathrm{E}[X] \to \mathrm{E}[T_f] \to \mathrm{E}[T_w] \to \mathrm{E}[Q]$ to get the different performance metrics, applying Little's law $N = \lambda T$ and $\mathrm{E}[T_f] = \mathrm{E}[T_w] + \frac{1}{\mu}$, we get the results shown in figure 4.69, where we compare negative exponentially distributed arrivals and holding times (on the left) with an assumedly quite realistic arrival process composed of negative exponentially distributed arrival instances at which bulks of packets with highly variable, $Lomax(2,1)$ distributed, bulk-size arrive simultaneously, $A = M^L$, in conjunction with the already often used, upper and lower bounded, bimodal holding time process, $B = \mathscr{B}eta(\frac{1}{3},\frac{2}{3},\frac{1}{2},2)$ modelling mostly short or long packet holding times (on the right).

The results on the left show that simulation and model fit each other, at least for the Markovian processes assumed. The mean throughput ϑ of an infinite system equals the arrival rate, here, for the

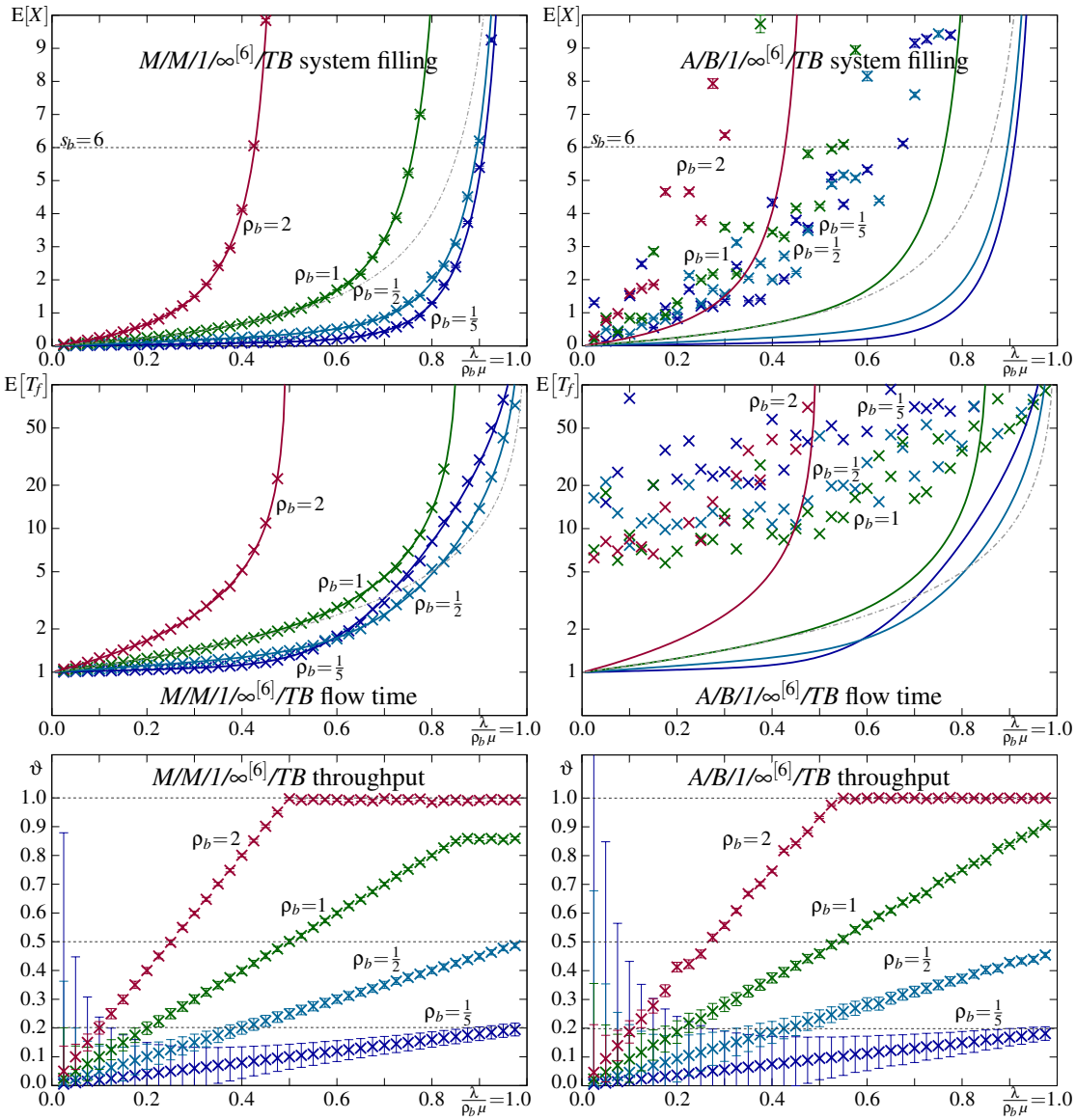


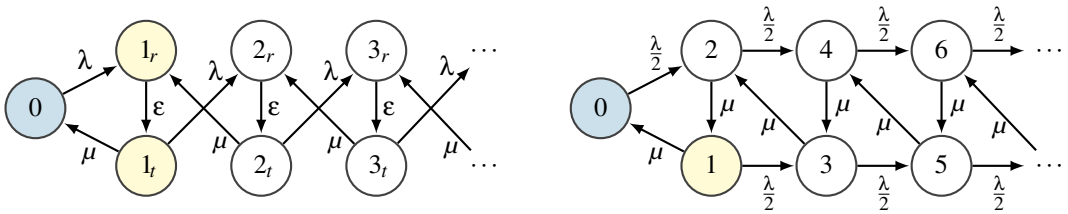
Figure 4.69: $A/B/1/\infty^{[6]}$ token bucket ingress controller, Markov arrival and holding times (left), negative exponentially distributed bulk arrivals with $Lomax(2,1)$ bulk size and $\mathscr{B}eta(\frac{1}{3},\frac{2}{3},\frac{1}{2},2)$ distributed holding time and token arrivals (right), infinite buffer $s=\infty$, bucket size $s_b=6$, $\mu=1$

token bucket controller, upper bounded by the token arrival rate, such that $\vartheta = \min\{\lambda, \lambda_b\}$. This hard boundary is perfectly approved by the simulation results, in both cases. Note also that we re-scaled the x-axis from the usual $\frac{\lambda}{\mu}=[0\,..\,1]$ to $\frac{\lambda}{\mu}=[0\,..\,\rho_b]$, where $\rho_b=\frac{\lambda_b}{\mu}$ is the virtual load from the token arrivals, being the mean service rate granted. Due to that, the throughput ϑ reaches the token rate λ_b at the right boundary of the diagram, if this is achievable. In case of $\lambda_b \geq \mu$ this is not possible, and thus the performance metrics approach infinity early.

Looking at the mean flow times $\mathrm{E}[T_f]$ on the left, also known as sojourn time, we recognise that we do not get the best performance for $\lambda_b=\mu$. In particular, the comparison to *M/M/1*, the dash-dotted curve, which represents the performance of a *leaky bucket limiter* with rate $\mu_b^{[LB]}=\lambda_b^{[TB]}$, reveals that for $\lambda_b<\mu$ the performance of the *token bucket controller* is considerably better at low loads, where short load bursts experience less delay. At high loads, in respect to the token-rate, the achieved mean flow time is inferior for both, low and high token-rates λ_b. Without prove, we conclude that the system performs best for a token-rate at around $\lambda_b=\frac{\mu}{2}$, and that this can be extended for multiple flows into $\sum\lambda_b=\frac{\mu}{2}$.

If we now turn to the results on the right, we recognise that the assumed arrival process with the negative exponentially distributed bulk arrivals with infinitely variant bulk sizes represents a really difficult load. Both, the system filling $\mathrm{E}[X]$ and the mean flow time $\mathrm{E}[T_f]$ are heavily increased by the simultaneous arrival of packets. In consequence, the performance is evidently inferior to that of better distributed arrivals. In addition, that the by default calculated confidence intervals do not fit any smooth curve reveals that the calculation of confidence intervals is based on the *central limit theorem*, which fails in case of infinite variance. However, the core functionality of the token bucket controller, the upper bound *mean throughput*, $\vartheta \leq \lambda_b$, is perfectly achieved. For bulk arrivals with rather smooth service the bound is only reached later, meaning at higher load in respect to the token rate. And again, the achievable maximum throughput in case of $\lambda_b = \mu$ is higher and here this token rate appears to be the optimum also in terms of the mean flow time $\mathrm{E}[T_f]$. This reminds us that what is best for Markovian processes may not be the best in case considerably different random processes are involved.

Concerning the integration of different stochastic processes in the model used to get numeric results, we note that the integration of any arrival or departure process that causes transitions to filling levels other than neighbouring levels, cannot be solved using the MGM in general. Only if joining filling levels into isolated and structurally identical *super*-levels is possible, such that no transitions bypassing a neighbouring *super*-level exist, the MGM can be adapted to analyse such a system. For example, we can integrate any *phase-type* arrival or service process where the filling level does not change until an event generating phase is left, as depicted by the left state transition diagram in figure 4.70. As well, we can level upper bounded bulk arrivals by joining as many states into a single *super*-level as load units can at once arrive within a bulk, as depicted by the right state transition diagram in figure 4.70. However, notice that the system filling is not unique within a *super*-level,



levels topology *levelled chordal topology*

Figure 4.70: MGM compatible state transition diagram (left) resulting from arrival and service processes interrupted by a common reception (ε) phase, and (right) resulting from levelling upper bounded bulk arrivals (shown for a deterministic bulk-size of two)

and that whenever from any state all other states can be reached, as for example with negative exponentially distributed bulk sizes, no *levels topology* can be found. In consequence, such a state transition diagram cannot be solved using the plain matrix geometric method. In contrast thereto, all finite systems can be solved using the *matrix analytic method* (MAM) if the calculation effort is tractable. Sophisticated methods to decompose the potentially huge multidimensional matrices into tractably dimensioned sub-matrices increase the calculation complexity but do not change the principal approach. These methods are exceptionally utile, but in general, applicable for very specific cases only.

Returning to the here analysed ingress control mechanisms we recognise that in contrast to leaky bucket limiting, the token bucket controller enables tokens to pile up in the tokens queue (bucket). Thus, short load peaks can pass the system at maximum rate, $\vartheta(t) = \mu$, whenever there are enough tokens in the bucket. This improves the response time for short messages commonly used for signalling and other time critical demands. However, if the arrival rate $\lambda(t)$ exceeds λ_b for a sufficiently long time, such that the tokens-bucket becomes depleted (no token left in the bucket), than

the mean departure rate equals the token rate, $\vartheta(t) = \lambda_b$, and the token bucket controller performs alike the leaky bucket limiter, until tokens accumulate in the bucket again.

Effectively, the finite bucket limits the maximum duration t_{burst} of the load peak that can pass the system at full rate μ, and thereby the maximum load volume $\ell_{burst} = t_{burst}\,\mu$ that may be served consecutively at full rate μ.

$$t_{burst} \leq \frac{s_b}{\mu - \lambda_b} \qquad \rightarrow \qquad \ell_{burst} \leq \frac{s_b}{1 - \frac{\lambda_b}{\mu}} \tag{4.28}$$

For $\lambda_b > \mu$ the leaky bucket controller becomes ineffective in terms of burst limiting because in that case tokens arrive more quickly than they are in average consumed. Thus, in that case the tokens-bucket never empties in average and infinitely long bursts are possible. Latter also occurs for the rather academic case with an infinite bucket $s_b = \infty$, here independent of λ_b but rather theoretically because an infinite number of tokens needs to accumulate in the bucket first.

The ingress smoothing effect of the discussed control mechanisms is not visible from the usual queueing system behaviour. To see the effect we show in figure 4.71 the arrival and departure coefficient of variation for leaky bucket and token bucket controlled ingress flows. The compared

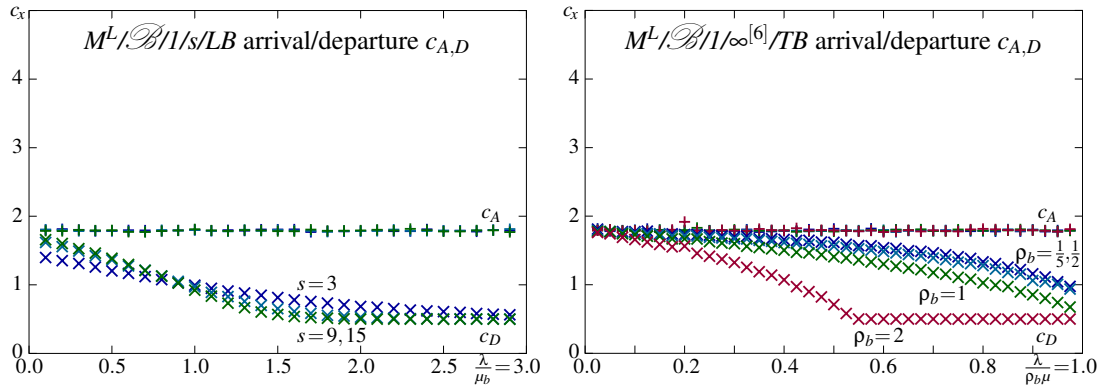


Figure 4.71: $M^L/\mathscr{B}/1/s$ leaky bucket and $M^L/\mathscr{B}/1/\infty^{[6]}$ token bucket ingress controllers' coefficient of variation for arrivals c_A (+) and departures c_D (×), with $\mu_b^{[LB]}=1$ and $\mu^{[TB]}=1$, respectively

systems are very different: the leaky bucket is by definition finite and we show results for bucket sizes $s=\{3|9|15\}$, whereas the token bucket controller has an infinite load queue and a finite tokens-bucket with size $s_b=6$ and is analysed for different token arrival rates $\frac{\lambda_b}{\mu}=\rho_b=\{\frac{1}{5}|\frac{1}{2}|1|2\}$. However, the applied arrival and service processes are the same for both. Considering mean arrival rates exceeding the set rate not practical, $\lambda<\mu_b,\lambda_b$ respectively, we restrict the comparison to the relevant x-axis ranges. Doing so, we recognise that the departure process of the leaky bucket limiter (left) is less varying than that of the token bucket controller (right). As already mentioned, the systems are quite different and this observation might be owed to the finite/infinite buffer assumption. However, by design the leaky bucket should be smoother because it lets no load pass at more than the set bucket rate $\mu_b \leq \mu$ at any time, in contrast to the token bucket controller where the passing of short bursts at full rate μ is a design feature.

Viewed from the customer (client/packet) perspective, such a *token bucket based ingress control scheme* performs comparable to a queueing system with server vacation, where the vacation intervals depend on the current load. This assumption is based on the fact that with token bucket control the departure stream contains idle periods even if the clients queue did not become empty. These *gaps* are caused by the the tokens-bucket, which empties with a likelihood that depends on the current system load. However, to use this observation for an alternative modelling approach is burdened by the complex relation between the system load and the tokens-bucket idle process, representing interdependent stochastic processes. Still, at source side the token bucket principle can even be

extended into a scheduling mechanism, as for example implemented in the *Linux* kernel since version 2.4.20, called *hierarchical token bucket* (HTB) mechanism [127]. In this study we are more interested in the network intrinsic mechanisms, the achievable service quality and and the fairness issues that may arise, as discussed individually with every presented scheme and more globally in section 4.3.4.

Ingress smoothing

An issue common to all queueing systems is the performance degradation due to the variance of arrivals. Therefore, it would be best to reduce the arrival coefficient of variation c_A to zero. In general this is possible only if we a priori know the precise mean arrival rate λ. Only than, we can design a *G/D/1* ingress controller that releases load units into the network equally spaced in time, where $\mu_{[D]}=\lambda_{[A]}$ needs to be set in advance in order to achieve the target. Not knowing the precise mean arrival rate in advance, we sketch a dynamic mechanism that should achieve at least some variance reduction also for loads with a priori unknown mean arrival rate. To achieve this, we use a leaky bucket limiter with an $i=X(t)$ dependent service rate μ_i, as shown in figure 4.72.

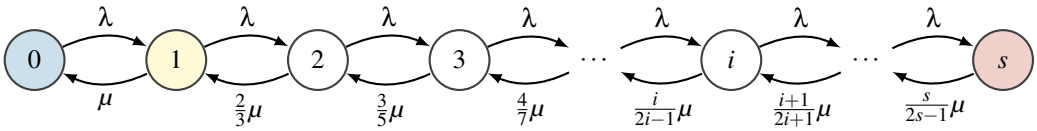


Figure 4.72: A smoothing leaky bucket limiter with $\mu_i=\frac{i}{2i-1}\mu$, which limits the ingress rate and $i=X(t)$ dependently spreads the departures of successively served load units over time

Solving the *system of differential equations*, $\dot{p}(t)=Q\,p(t)$, defined by the state transition diagram shown in figure 4.72, using the algorithms introduced in section 3.2.1, we can calculate the distribution of the flow time $F(T_f)$ shown in figure 4.73. The smoothed flow time distribution $F(T_f)$ at $\lambda=\frac{\mu}{2}$

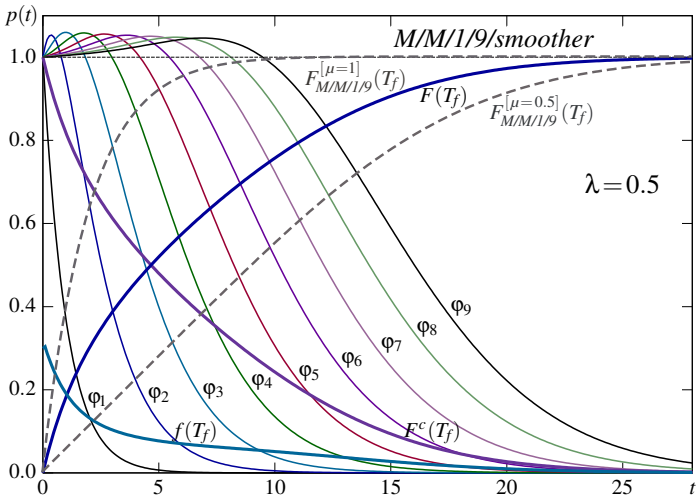


Figure 4.73: Flow time distribution $F(T_f)$ of the smoother sketched in figure 4.72 for system size $s=9$, $\lambda=0.5$, $\mu=1$ compared to *M/M/1/9* flow time *cdf*s at same λ and $\mu=\{1|0.5\}$ respectively

is quite curved and falls nicely in between the flow time cdf's of a simple *M/M/1/s*-system with service rates $\mu=1$ and $\mu=\frac{1}{2}$. Note, to correctly sum-up the F^c-components φ_i of different system entry states, we need to consider both, the arrivals lost due to blocking $\lambda(1-P_b)$ because these load share does not contribute to the $F(T_f)$ experienced by leaving clients, as well as the effective virtual service rate $\mu^*=p_0\mu+\sum_{i=1}^{s}p_i\mu_i$, which obviously depends on the system filling, and thus, decreases with increasing system load.

If all ingress limiters of all flows operate independently and unsynchronised, this should yield smoothed packet arrivals at network nodes, independent if bundled into shared LSPs or not. Thereby, the mean queue filling at downstream nodes is assumedly reduced and thus the accumulated waiting time along paths as well. However, the waiting time in the ingress limiter (smoother) is increased, and the maximum ingress rate supported drops to $\frac{s}{2s-1}\mu$. Also, the point by point calculated flow time distribution shown in figure 4.73 does not provide much help toward the departure characteristic gained, which would be required to assess any advantage on downstream nodes.

Basically, *smoothing* may as well be included at every egress port of network nodes in order to smooth the ingress to the next node. However, any evaluation of possible benefits demands the evaluation of queueing system chains, as outlined in section 4.3, because any change of the traffic (flow) characteristic becomes effective at downstream nodes only.

Windowed ingress control

Another approach to ingress control is limiting the amount of traffic c_w inserted within a certain time-span τ_w, called *windowing*. Let us assume this is implemented with a τ_w-discrete clock: as long as the load contingent c_w allowed per clock cycle τ_w is not exhausted, traffic is served at full rate μ. If c_w is consumed before the clock cycle ended, no traffic is served, all arrivals become buffered in the queue and no load is served until the next clock-cycle starts. For every cycle the traffic contingent allowed to be transmitted is re-set to c_w, unused contingents of a cycle are not carried over to the next cycle. This assures that the maximum load per cycle is well bounded, such that $\vartheta_w \leq \frac{c_w}{\tau_w}$ is assured.

To model this behaviour we assume a queueing system comprising two queues: an outer queue to buffer any load that cannot be served in the current window and an inner queue buffering the load that is served in the current window. In between these two queues is a switch controlled by a load-counter that limits the amount of traffic $\sum_{\tau_w} \ell_{in}$ forwarded within each cycle of the τ_w-clock. This system is sketched in figure 4.74. Note, if at the begin of a cycle less than c_w load is waiting in the outer

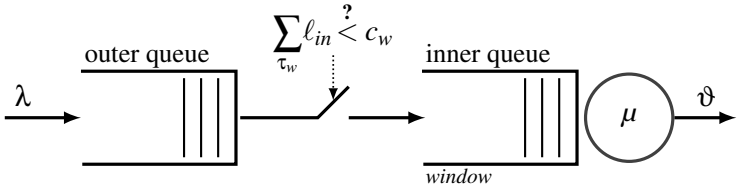


Figure 4.74: Windowed ingress controller that limits the mean ingress rate to $\vartheta \leq \frac{c_w}{\tau_w}$

queue, the switch controlled by the counter remains closed until the contingent is consumed. The arrival process to the inner queue is thus a composite of (i) τ_w interleaved batch arrivals, (ii) regular individual packet arrivals, and (iii) no arrivals while the switch is opened. Due to the controlled ingress to the inner queue its filling never exceeds c_w. The outer queue can be assumed infinite or finite, depending on whether losses shall be foreseen or not.

If windowing is implemented differentially, $\tau_w \to 0$, it becomes the leaky bucket limiter with rate $\mu_b = \frac{c_w}{\tau_w}$. This is evidently more efficient than the time discrete mechanism. Therefore, we skip any evaluation of the inferior system based on the two-dimensional state transition diagram that results for the queueing system shown in figure 4.74. However, the *windowing* approach provides two parameters to dynamically adjust the mean ingress load that a source may release into the network, the window length in time τ_w and the window size in load units c_w, which to some extend can be used to control the variance of the inserted flow (high for large τ_w, low for small c_w).

The most common ingress controller based on this paradigm is TCP. Its basic version limits the number of packets released into the network still awaiting acknowledgement to less than the currently set window size c_w. Whenever acknowledgement for some packets is received the according number of packets may again be transmitted. Therefore, the clock τ_w is replaced by a feedback

mechanism intrinsic to the transport protocol. The length of the window is therefore determined by the connection dependent and network load modulated round-trip-time, τ_w =RTT, a random variable, and the re-adjustment of the transmission allowance is spread across the window duration, with grant re-adjustment amounts dependent on the acknowledgement scheme used: single, sequence or group of packets acknowledged by a single acknowledgement message, again potentially a random number. Basically, this realisation represents an *on-top* approach that requires the cooperation of end systems, being the source and destination of flows. In contrast to the mechanism presented until here, TCP cannot be implemented within the network layer or stand-alone to control aggregate loads.

That the size of the window c_w is in addition adjusted dynamically represents an extension that enables the mechanism to adopt itself autonomously to changing network conditions. Actually, this is the core feature of the TCP/IP bundle because as already mentioned, IP is not stable without TCP. The ingress limiting is only a necessary side effect, and in general, all TCP sources are greedy in that they always try to get the maximum transmission rate at any time, not considering any concerns on actual demand, network sanity, or fairness.

The literature contains a plethora of proposals on how to assess and optimise the performance of such systems due to TCPs huge popularity. Therefore, we sketch in figure 4.75 the main mechanism only, not including the start-up phase or the time-out procedure, and refer the kind reader to the very rich literature on the many TCP variants for a more detailed evaluation.

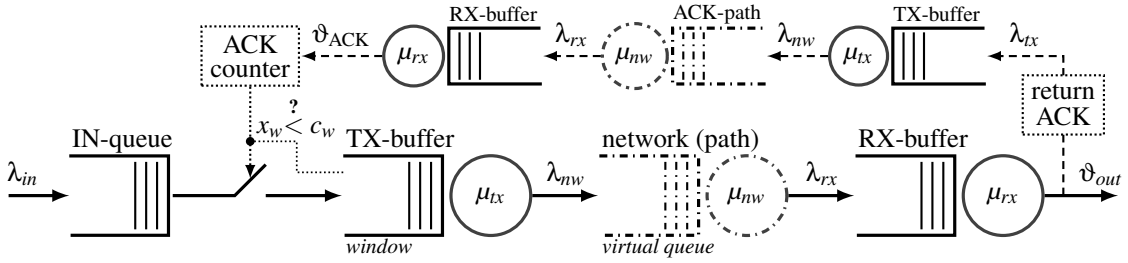


Figure 4.75: Acknowledgement controlled windowing (TCP in regular operation)

Looking at figure 4.75 it becomes apparent that modelling such a feedback based ingress control mechanism demands to solve a queueing network, precisely a cycle of queueing systems, with commonly far from Markovian holding times per node. Thus, the performance of such an ingress controller heavily depends on the current network condition.

First, note that the forward path of the load and the return path of the acknowledgement messages (ACK) are independent paths, which may share the same nodes in reverse order, but need not and for sure do not face the same resource utilisation. If the ACK-messages are transmitted as signalling packets, they may be served privileged, if piggybacked in the header of regular load retuned to the source (bidirectional path) than they are delayed at intermediate queues alike the load on the forward path. Still, the utilization of buffers is likely different because commonly the load is not equally heavy in both directions, and thus, the delays of forward and backward path may differ. Note, the switch in front of the transmit buffer is initially closed until the allowed number of load units awaiting acknowledgement c_w has entered. After this phase the switch is operated by the received ACK-messages, letting in exactly as many load units as currently being acknowledged.

Second, modelling the path across an IP-network is an open issue because to do this we need to know also the amount of competing load at each node, which itself may be controlled by the same or a similar ingress control mechanism. Sharing the same resources causes correlation of the control mechanisms as discussed in the end of section 4.2.2.

Third, the model sketched does not consider packet losses; neither on the forward path nor on the backward path. In practice these are essential because they are used to identify congestion and trigger the consequential reduction of the ingress rate, $c_w \rightarrow c_w(t)$.

Even though the model does not consider all aspects of TCP, we my draw some conclusions from the simplified model, leading to a further simplified model. Let us therefore assume that the ingress rate per flow λ_{in} is far below the access and network capacity μ_{tx}, μ_{nw}, latter being the minimum capacity along the used path, and also that some *network management* assures that the queues at network nodes are only sparsely filled, meaning a QoS friendly low utilization $\rho_{nw} \ll 1$ is maintained, than we may postulate that the mean waiting time at network resources is only a fraction of the holding time thereof, and thus that the main latency of packets belonging to a flow occur in the ingress queue (IN-queue). In consequence, ignoring all latencies added by network resources, we may simplify the model and approximate the entire send-ACK-loop by a single server with capacity

$$\mu_{\text{TCP}}(t) = \frac{c_w(t)}{\text{RTT}}. \tag{4.29}$$

Based thereon we can model the entire TCP connection by a single *GI/G/1* queueing system, given that we can calculate the distribution function that results for the division of the two random variables $c_w(t)$ and RTT. For the current *load of a TCP connection* we get

$$\rho_{\text{TCP}}(t) \approx \frac{\lambda_{\text{in}}}{\frac{c_w(t)}{\text{RTT}}} = \frac{\lambda_{\text{in}}}{c_w(t)}\,\text{RTT} \tag{4.30}$$

and notice the RTT-*product*: the longer the connection is in terms of time, the higher becomes its virtual load $\rho_{\text{TCP}}(t)$. Vice versa, the bigger the window c_w is, the lower becomes the virtual load.

Having simplified the problem that far from reality, we see no reason not to ignore the influence of different distributions as well, and thus assume Markovian arrival and service processes. Using the known results for *M/M/1* we can assess a rough estimate on the performance, in particular the *sojourn time* representing the mean end-to-end flow time of a TCP connection at different c_w and RTT,

$$\text{E}[T_{f,\text{TCP}}](t) \approx \frac{1}{\mu_{\text{TCP}}(t) - \lambda_{\text{in}}} = \frac{\text{RTT}}{c_w(t) - \lambda_{\text{in}}\,\text{RTT}} \tag{4.31}$$

where due to assuming an infinite IN-queue we need $\lambda_{\text{in}}\,\text{RTT} < c_w(t)$ for stability reasons. In practice this is rather irrelevant because both, the maximum amount of data that may arrive as well as the available memory provided to queue traffic, are always finite.

Concerning practice, neither Markovian arrivals nor Markovian holding times are realistic. To correctly dimension the IN-queue, we need to assume batch arrivals to consider the presentation layer where files are split into batches of packets. Also the files may be requested in batches, as it for example occurs with framed web-pages, where each frame represents a file that is downloaded individually. For a better model of the holding time we should on one side consider the deterministic propagation delay component, which results from the finite speed of light and path dependently lower binds the holding time of a connection across the network, and on the other side the maximum size of packets, which upper binds the holding time component introduced by the least line rate $r_{tx} = \frac{\mu_{tx}}{\text{E}[\ell_{packet}]}$ at which the bits of a packet are put on the journey one by one. In contrast thereto, ignoring the distribution of the RTT and assuming it to be constant seems less critical because for the simple single server model we already had to assume negligible load at all network nodes. Thus, if we assuming an RTT that only considers the constant propagation and processing delays along forward and backward paths, and add a small fraction thereof to consider the queueing delays at intermediate nodes, the introduced error should not be too big.

However, never should we assume that the absolute results gained from a such heavily simplified model are reliable. Results gained form such simple models only serve well for the comparison of variants, when the same inaccuracy can be assumed for all variants evaluated. Real measurement and more realistic simulation of data transmission cannot be replaced by such a poor model when it comes to service level agreements and other vital assets.

4.3 End-to-end performance

In practice the mechanisms presented in the previous sections are not isolated. They perform their individual tasks in conjunction with other systems that also alter the traffic streams' characteristics. Thus, in general these systems influence each other [128]. In consequence we would in general require a global solution that considers all influence factors to asses the actual end-to-end quality of load transportation across communication networks. According to the *butterfly effect*, a change in one place may cause consequences on timely offset distant flows never passing the location where the change actually occured.

A major problem in analysing meshes composed of per se already rather complex systems is also the plurality of parameters: every mechanism can be individually parametrised, opening a plethora of evaluation options. On the other side, results achieved for a particular setting cannot be used to assess the quality of other settings. A brief survey of queueing network solution and approximation approaches than convinced us that a joint analysis of entire networks of queueing systems is out of reach. In the sense of *leave expert problems to the relevant experts*, we no further address the topic and drop the intention to model chains of complex queueing systems carrying multiple flows of test and background traffic, as sketched in figure 1.33.

However, to round up the topic we briefly introduce in section 4.3.2 the basic methods to solve some queueing networks, and in section 4.3.3 how to approximate more general networks of queueing systems. Finally, in section 4.3.4 the underlying issue validating all the effort to model/simulate system behaviours, the *quality of service* (QoS) provisioning problem is formulated, laying the bridge to the topic finally addressed in section 4.4.

4.3.1 Chained queueing systems

Systems comprising two simple queueing systems in series, so called *tandem systems*, have been modelled and analysed for long. The results can be found in most textbooks, for example [34, chapter 15], together with the later on outlined methods on solving different queueing networks.

A simple tandem system composed of two queues with different mean service rates μ_1, μ_2 is sketched in figure 4.76 together with the state transition diagram representing it. Note, when we limit

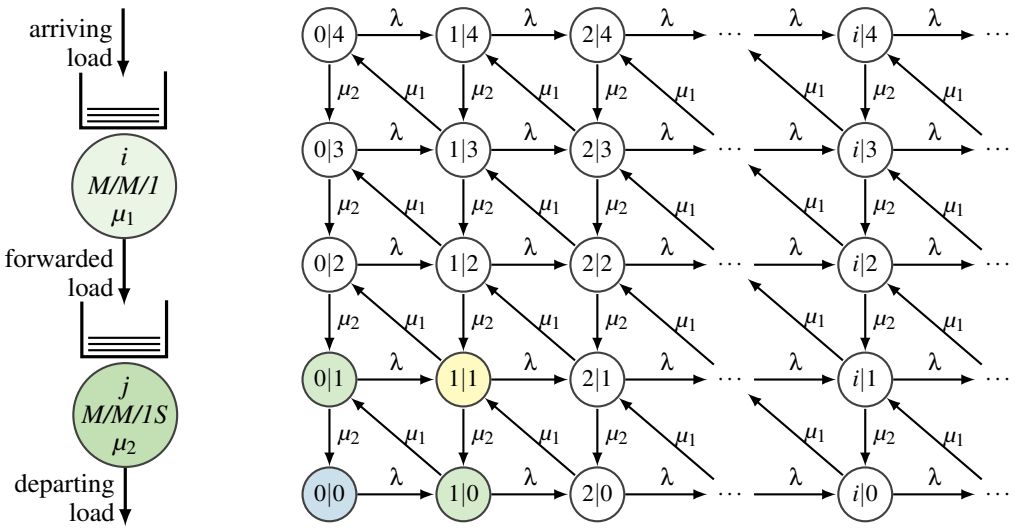


Figure 4.76: Two *M/M/1* systems in tandem with back-pressure and state transition diagram, state indices $(i|j)$ indicating number of jobs present per system, $s_1 = \infty$, $s_2 = 4$

the size of the second system, as done for the depicted example, than we need to precisely define

what blocking causes. Here we assume that a full queue at the second system hinders the client to leave the first system, putting the serving there on halt, which is commonly called *back-pressure*.

The stand alone state transition diagram for a solitaire *M/M/1* system is one dimensional. Two such systems yield a two dimensional state transition diagram, and every additional *M/M/1* system in series adds another dimension. If the chained systems themselves need multiple dimensions to be modelled by a state transition diagram, then the number of state transition diagram dimensions of the chain of systems is the *sum over the individual system's dimensions*. Drawing such complex state transition diagrams can be cumbersome. It is better to define a conclusive rules-set and based thereon compose the Q-matrix to represent the state transition diagram mathematically, instead of attempting to depict a multi-dimensional graph by two dimensional views.

On the other side, if the arrival time or service time of a system can be modelled by a phase type distribution, each state in figure 4.76 needs to be extended by the Markov phases the process is composed of. In consequence, state transition diagrams of chained systems quickly become hard to grasp and any errors therein, in particular incorrect or missed Q-matrix composition rules, cause incorrect results. If the errors primarily cause incorrect transitions out of states with small probabilities, then the divergence of analysis results may be subtle and easily missed.

However, if only one queue is infinite we recognise a repeating levels structure in the dimension of the infinite queue and can apply the *matrix geometric method* (MGM) to get the state probabilities $p_{i,j}$. From these we can calculate both, the individual mean queue fillings

$$\mathrm{E}[X_1] = \sum i\, p_{i,j} \qquad \mathrm{E}[X_2] = \sum j\, p_{i,j} \quad \rightarrow \quad \mathrm{E}[X] = \sum (i+j)\, p_{i,j} = \mathrm{E}[X_1] + \mathrm{E}[X_2] \tag{4.32}$$

as well as the mean number of customers residing somewhere in the chain. Using Little's law $N = \lambda T$ we get the mean flow times per system

$$\mathrm{E}[T_{f_1}] = \frac{\mathrm{E}[X_1]}{\lambda} \qquad \mathrm{E}[T_{f_2}] = \frac{\mathrm{E}[X_2]}{\lambda - \delta_2} \quad \rightarrow \quad \mathrm{E}[T_f] = \mathrm{E}[T_{f_1}] + \mathrm{E}[T_{f_2}] = \frac{\mathrm{E}[X]}{\lambda} \tag{4.33}$$

and their sum states the sojourn time of the chain, which can as well be calculated applying Little's law $N = \lambda T$ on the total mean system filling $\mathrm{E}[X]$. However, latter only applies with *back-pressure*, where losses δ_j in between the chained systems cannot occur.

In case all queues are finite we may even get the global flow time distribution. The basic transitions cycle, the exit to exit sequence $\lambda \rightarrow \mu_1 \rightarrow \mu_2$, is commonly best seen around the idle state. This cycle applies for any customer that arrives to the idle system, if we assume FIFO queueing or when no other customer arrived during the sojourn time of that customer. Its flow time is then the sum of all service processes because it never needs to wait in any queue as there is never any customer in front.

To get the flow time distribution we need the weighted sum over all possible conditional flow time distributions along any possible entry to absorption path in the state flow diagram depicted in figure 4.77. For FIFO queueing we just need to drop all arrival transitions to get the flow diagram from the state transition diagram. The transitions of the state flow diagram define the flow matrix Q_f, and solving $\dot{p} = Q_f\, p$ yields the conditional flow times from all possible entry state to absorbing state transitions. If we multiply these with the according entry probabilities, where the sum over all entry probabilities must be one as we only count served customers to get the flow time, and add-up the resultant curves, we get the global flow time distribution.

The state transition diagram of the example tandem system is very similar to the state transition diagram analysed in section 4.2.3. Thus it should be clear how to use the methods introduced in section 1.4.3 to solve the system of linear equations to get the state probabilities $p_{i,j}$, as well as the methods presented in section 3.2.2 to solve the system of differential equations in order to get the conditional flow time distribution functions $\varphi_{i,j}$ for the different states.

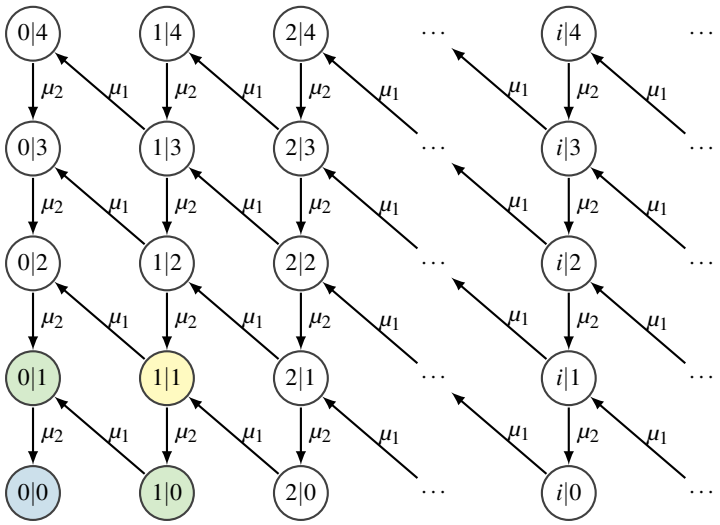


Figure 4.77: State flow diagram for two *M/M/1* systems in tandem with back-pressure, state indices $(i|j)$ indicating number of customers present per system, $s_1 = \infty$, $s_2 = 4$

Combined multi-flow control

Concerning telecommunication network nodes we commonly find more complex queueing systems. First we have the inevitable *transmit queue* of the output port, where packets become converted into transportable units of the next lower layer. In front of that we can integrate different scheduling policies and load limiting mechanisms, as for example sketched in figure 4.78. This queueing

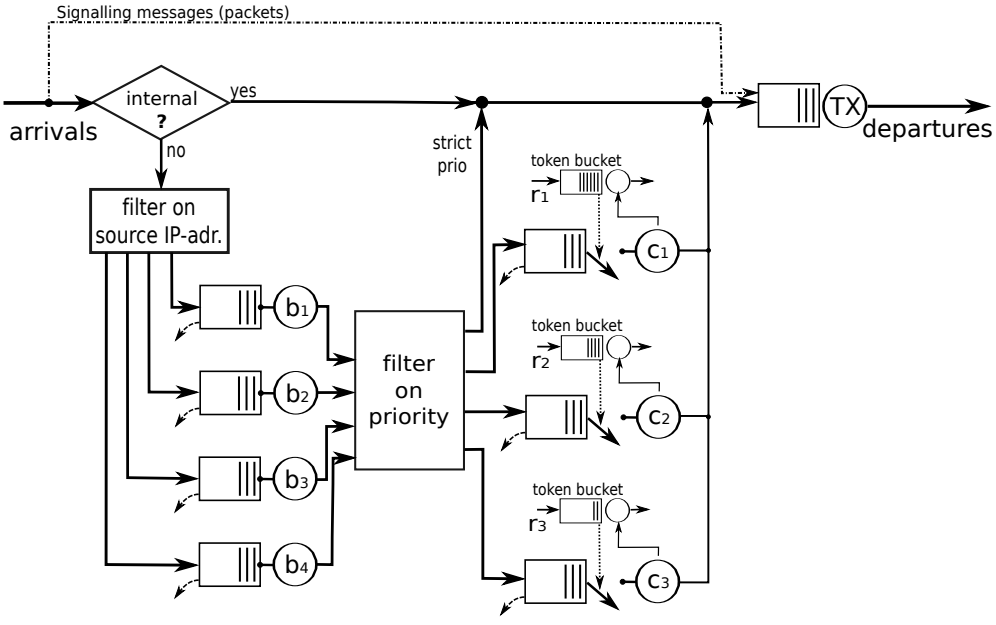


Figure 4.78: Ingress load management example as potentially implemented at an edge node

architecture tries to model the functionalities of an access node where the flows of different customers enter the network. A selection by the source IP-address will hardly be feasible at core routers where thousands of flows may be enclosed in the passing traffic aggregate. However, address masks and LSP-label and *class-of-service* CoS header fields may still be used at core network nodes for load separation, as for example sketched in figure 4.79. Here the prime task will be QoS differentiation.

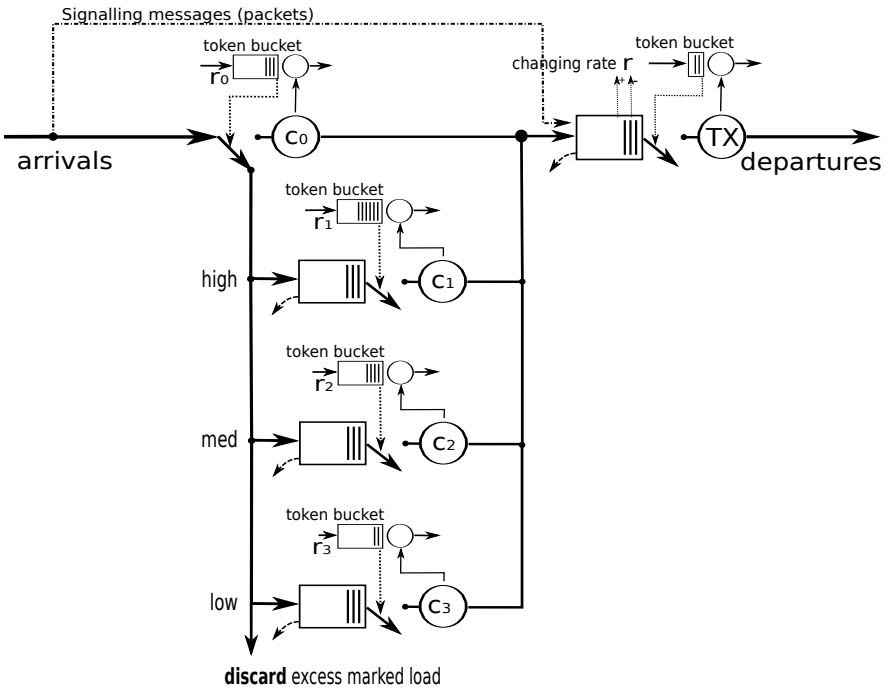


Figure 4.79: Trunk sharing and aggregate flow smoothing as it might be implemented at core nodes

However, the transmit queue at the interface to the lower layer may be configured as well. In the example shown in figure 4.79 we assumed a queue-filling controlled processing speed, which for example may be used to achieve distributed smoothing of the load distribution over time introduced in section 4.2.3. Note, for implementation consistency we assumed all sub-systems to be token bucket controlled queuing system, remembering that for bucket size zero it becomes a leaky bucket controller and for infinite token rate a regular *M/M/1/s* queueing system.

Clearly, state transition diagrams representing such complex tandem systems are hard to depict as two dimensional graphs. On the other hand, having shown in the previous sections how the individual systems can be represented by state transition diagrams, and in figure 4.76 how to combine the state transition diagrams of chained queues, it is also clear how these systems can be analysed using state transition diagrams.

Without analysis we can roughly state some configuration rules [129]: (a) if $\sum_i r_i < \mathrm{TX}$ the transmit queue cannot become persistently overloaded, and (b) if in addition $c_i < \mathrm{TX}\ \forall_i$ the departing traffic aggregate cannot consist of an uninterrupted sequence of load units belonging to the same traffic class. Concerning the operational tasks of the sketched examples we recognise that the example shown in figure 4.78 is likely found at edge nodes. The first stage prunes all traffic that exceeds the SLA a customer paid for, while the second stage implements flow differentiation based on priorities assigned to conform to the requested QoS. Note, the second stage may as well use custom based queueing (CBR) instead. The internal traffic bypassed refers here to fixed line VoIP traffic, which typically is an add-on service with a capacity demand negligible compared to streaming and download services. However, due to the stochastic occurrence of flows and their propagation along different paths at different times, flow management at edge nodes alone cannot reliably prevent critical network states at all times. This is the task of the tandem shown in figure 4.79, which primarily implements QoS differentiation. The bypassed signalling messages should in general constitute uncritical load volumes, and the self-maintained smoothing at the output provides an option to manipulate the departure distribution. Alternative approaches to these tasks are presented in [130].

4.3.2 Networks of queueing systems

If the provided *service* comprises the passage through several queueing systems, then the *quality of service* (QoS) is defined as the performance of the entire chain of systems passed, and not via the individual systems' performances. Evidently, a chain cannot be better then the weakest link, and more challenging, if all links contribute the same performance, the chain will in general be less well performing because the weaknesses of the individual systems can accumulate. In some situations this is advantageous, for example provides a chain of identical frequency filters a steeper filtering curve, but the losses, being the attenuation in case of frequency filters, accumulate unfavourably. For queueing systems latter applies for both, the losses and the time spent in the chain.

To correctly predict the *end-to-end* performance of a chain of queuing systems we need to model the entire chain because the departures from a system are in general not distributed alike the arrivals to the system. The problem of queueing chains is not restricted to packet switched networks. For example, it appears likewise with manufacturing lines, meshed transportation systems, and street networks. It also applies to sub-systems of network nodes, for example the chain composed of the line card receiving a packet, the router engine looking for the correct output port, and the transmit queue of the output port.

In the following we will commonly assume a more abstract network definition, where each node represents a single queueing system, not a network thereof as just exemplified. Only if the receivers and the routing engines would not contribute any delays, meaning if they could operate at infinite speed, this would equal the topology of a communication network as it is commonly shown. For simplicity and without loss of methodical applicability we will commonly not consider which task a queueing system performs. Considering latter would restrict the evaluation to specific node architectures only. In figure 4.80 we sketch such an abstract queuing network with an exemplary highlighted path, $1 \to 2 \to 4 \to 7$ (bold). Note that the backward path needs to pass either node 2

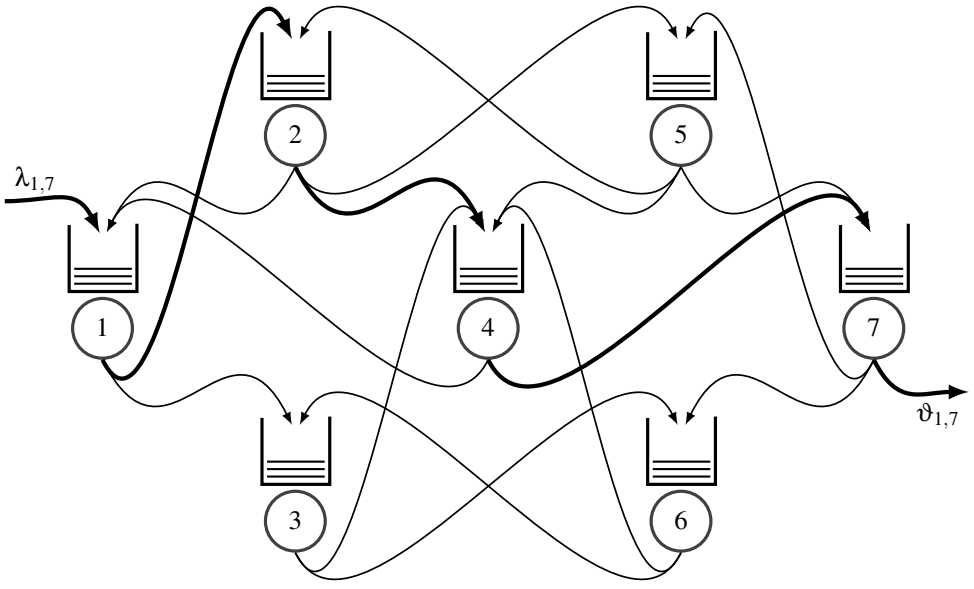


Figure 4.80: Small network of queueing systems with path $1 \to 2 \to 4 \to 7$ highlighted.

or 4 and has to include either node 5 or 6, which are not part of the forward path. Actually, there exist two alternative shortest paths, and for one of these a reverse path utilising the same nodes in reverse order exists. However, if bidirectional paths exist or not is of no relevance here. The very special and restricted scenario of solely bidirectional paths can always be extended to pairs of oppositely oriented unidirectional paths, if at all required.

Form the evaluation of individual queueing systems we know that the arrival distribution influences the performance of a queueing system. For chained systems, where the departure distributions form upstream systems define the arrival distribution to a downstream system, the departure distribution is very relevant for the end-to-end performance.

In particular with *packet switched networks*, where *cascades of queueing systems* are almost everywhere around to *provide the essential data transmission functionality*, we cannot neglect the fact that the departure process characteristic, being the inter-departure time distribution, is in general load dependent. Only for Poisson distributed arrivals (negative exponentially distributed inter-arrival times), infinite queues, and negative exponentially distributed service times, and some other very special infinite systems, is the departure process again a Poisson process with negative exponentially distributed inter-departure times and load independent variance. In general are the characteristics of the departure process, meaning its variance σ_D^2 and moments above, dependent on the arrival and service process properties, A and S, as well as the current system load ϱ.

$$D = f(A, S, \varrho = \tfrac{\lambda}{\mu})$$

In [131, 1956] P. J. Burke first proved that the departures of an *M/M/n* queueing system is again negative exponentially distributed (Poissonian). An alternative prove was provided by E. Reich in the begin of following year [132], just before the thereon based paper of James R. Jackson [133] became published. An intuitive prove is based on the reversibility of the M/M/1 queueing process [134]. Notably, *Burke's theorem* is valid for few cases only, and it is assumed that Burke's theorem does not extend to queues with a Markovian arrival processes (MAP): the output process of a *MAP/M/1* queue is a MAP only if the queue is an *M/M/1* queue [135].

Generally, we know the mean departure rate being the mean system *throughput* ϑ. Thus, we may also calculate the mean inter-departure time

$$\vartheta = \lambda - \delta \qquad \rightarrow \qquad \mathrm{E}[T_D] = \frac{1}{\lambda - \delta}, \tag{4.34}$$

where $\delta = \lambda P_b$ is the *blocking rate*. For loss-less systems is $\delta = 0$, such that their average throughput equals the average arrival rate. This property can be used to quickly assess the mean load present at chained queueing systems. However, in general it does not provide any information about the inter-arrival time distribution present at downstream systems and is thus not sufficient to assess the performance of queueing system cascades.

In section 4.3.1 we assumed that all processed load form an upstream system is forwarded to a downstream system. The smart feature of meshed networks is that this is not the case. The load is distributed, meaning that it follows defined paths, commonly chosen such that it passes the least number of intermediate systems to minimise the total processing effort. For the service quality this may not be the best choice, to optimise the performance it may be advisable to intelligently distribute the load among the available resources. Anyhow, if we may assuming that all flows are composed of independent and identically distributed packets, being the network customers getting queued and forwarded node by node, we may assume constant splitting factors to model the routing of flows. Given the traffic matrix and the routing tables we can calculate the planned mean link loads and thereby the splitting factors (*probabilistic routing*) that apply for each network node individually. This probabilistic routing defines (a) how the ingress load to each node is randomly composed of departures from neighbouring nodes and the outside, and (b) how the egress load is randomly forwarded to subsequent nodes or leaves to the outside, as shown in figure 4.80.

Note, networks with no outside are called closed networks, and every open network can be closed by adding infinite queueing systems per ingress-egress pair, which serves as ingress generator and load unit buffer, accepting the according departing flows, as shown in figure 4.81. To model an Erlang setting we assume an outside system that state independently generates arrivals to the open

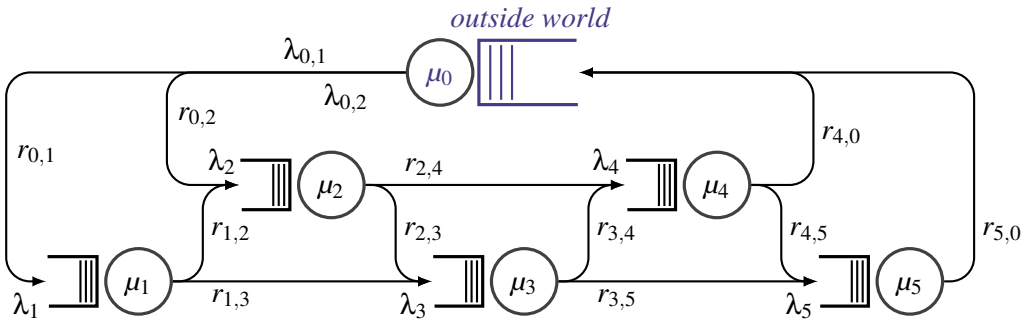


Figure 4.81: Closing an open queueing network by a queueing system that models the outside

network with the rate $\mu_0 = \sum_i \lambda_{0,i}$. To get an Engset setting, where the arrival rate λ is defined per customer, we initially assume the outside queue to be filled with the finite number of customers specified for the setting, and assume a state dependent arrival generation with rate $\mu_0 = x_0(t)\,\lambda$. In any setting, the queue that models the outside collects all traffic departing the open network, being egress and lost traffic.

Note that by closing the network the number of customers residing in the queues of the closed network, including the outside node, must be constant. Based on that, we may define the *network load* ρ_{nw} as the relation of the mean number of customers in the open network to the total number of customers $c = \sum_{i=0}^{N} \mathrm{E}[X_i]$ of the closed network.

$$\rho_{nw} = \frac{\sum_{i=1}^{N} \mathrm{E}[X_i]}{\sum_{i=0}^{N} \mathrm{E}[X_i]} = \frac{c - \mathrm{E}[X_0]}{c} = 1 - \tfrac{1}{c}\,\mathrm{E}[X_0] \tag{4.35}$$

Randomly splitting the departure flow of a node using constant factors may not perfectly fit reality, where commonly the traffic is forwarded according to per flow routing tables such that the load units of each flow remain together. However, only with random splitting (Markovian routing) we may assume that the distribution of the load forwarded to the neighbour node j, $\lambda_{j,i}$, equals the distribution of the load aggregate leaving node i, ϑ_i, multiplied by the splitting factor $r_{i,j}$, which represents the routing probability, for which $\sum_i r_{i,j} = 1$ is by definition fulfilled.

$$\lambda_{j,i} = r_{i,j}\,\vartheta_i \qquad \text{and} \qquad \sigma^2_{\lambda_{j,i}} = r^2_{i,j}\sigma^2_{\vartheta_i} \tag{4.36}$$

Having closed the network by a node 0 representing the outside and presuming infinite queueing systems throughout the network, such that $\vartheta_i = \lambda_i\ \forall_i$, we can calculate the load at each node by solving the linear equations system

$$\vec{\lambda}_{in} = \underline{R}\vec{\lambda} + \vec{\lambda}_{out} \tag{4.37}$$

where the matrix $\underline{R}$ holds the routing probabilities $r_{i,j}$. The equality of this matrix equation is governed by the *flow conservation law*: the mean flow entering a node must equal the mean flow leaving the node. The arrivals from the outside are given, $\lambda_{in,i} = \lambda_{0,j}$, and the departures to the outside result from $r_{i,0} = 1 - \sum_{j=1}^{N} r_{i,j}$ and $\lambda_{out,i} = r_{i,0}\,\lambda_i$. The resultant vector components of $\vec{\lambda}$ then yield the steady state mean arrival rates λ_i per node, from which each node's state probabilities $p_{i,k}$ can be calculated individually. These constitute an $\infty \times N$ matrix P that precisely describes the network's detailed state according to a joint state transition diagram at the given load $\vec{\lambda}_{in}$.

Henceforth we use in this section *network state probabilities* defined as the probability that each system hosts a particular number of customers $x_j = i_j$. To clearly separate system filling

probabilities $p_{i,k}$ from these, we use $\pi_{\vec{x}}$ to identify these *network state probabilities* and refer to the *filling pattern* $\vec{x}$ as *network state*.

$$\pi_{\vec{x}} = P[x_1{=}i_1, x_2{=}i_2, x_3{=}i_3, \ldots, x_N{=}i_N] = \prod_{j=1}^{N} p_{x_j,j} \tag{4.38}$$

Feed-forward networks

The methods found for chained systems can be extended to so called *feed-forward networks*, where all queues can be arranged such that all customer flows occur in one direction only. The open network example shown in figure 4.81 is such a feed-forward network. Evidently, closed networks cannot be feed-forward networks, whereas closing a feed-forward network might be used to analyse it.

In case of two *M/M/1* queueing systems in tandem we get

$$\pi_{\vec{x}} = P[X_1(t){=}i_1, X_2(t){=}i_2] = \rho_1^{i_1}(1-\rho_1)\,\rho_2^{i_2}(1-\rho_2) \tag{4.39}$$

$$\mathrm{E}[X_1] = \sum_{\vec{x}} i_1\,\pi_{\vec{x}} \quad \rightarrow \quad \mathrm{E}[T_{f,1}] = \frac{\mathrm{E}[X_1]}{\lambda} \tag{4.40}$$

$$\mathrm{E}[X_2] = \sum_{\vec{x}} i_2\,\pi_{\vec{x}} \quad \rightarrow \quad \mathrm{E}[T_{f,2}] = \frac{\mathrm{E}[X_2]}{\lambda} \tag{4.41}$$

$$\mathrm{E}[T_{f,chain}] = \mathrm{E}[T_{f,1}] + \mathrm{E}[T_{f,1}] = \frac{\mathrm{E}[X_1] + \mathrm{E}[X_2]}{\lambda} = \frac{1}{\lambda}\sum_{\vec{x}}(i_1 + i_2)\,\pi_{\vec{x}} \tag{4.42}$$

where the result for $\pi_{\vec{x}}$ is of product form. Evidently, this approach yields the same results as found with the joint state transition diagram presented in section 4.3.1. The last equation also conforms that in case of no losses Little's law $N = \lambda T$ is applicable for networks of queues as well.

Please refer to the literature for more details, for example [34]. Broadcast networks are typical examples of the feed-forward network type. Communication networks are in principle no feed-forward networks because communication demands a return path. However, transmission channels, in particular multicast channels, and entire lower layer or overlay networks with special topologies, for example *content distribution networks* (CDN) and *tree topologies* in general, mostly fulfil the requirements to be modelled as feed-forward network. Networks with feedback cannot be modelled as feed-forward networks.

Jackson networks

Based on the surprisingly simple *product form* of the results found for chained *M/M/n* queues, where customers visits each queue in order, James R. Jackson found that any network of *M/M/n* queueing systems shows this property, if only the arrivals from the outside to the network are all negative exponentially distributed, the servers have negative exponentially distributed holding times, and all traffic loads are Markovian (probabilistic) routed according to a common static routing matrix R.

We note that routing loops and $r_{i,i} \geq 0$ are not prohibited, and thus feedback is allowed with Jackson networks [133]. This is the major extension compared to feed-forward networks and states that *M/M/n* systems within Jackson networks perform as if all arrivals to every node of the network would conform to a Poisson process and that the individual system fillings $\mathrm{E}[X_i]$ are thus independent of each other.

The network state probabilities $\pi_{\vec{x}}$ of these networks then can be calculated from

$$\pi_{\vec{x}} = \mathrm{P}[i_1, i_2, .., i_j, .., i_m] = \prod_{j=1}^{m} \mathrm{P}[X_j(t){=}i_j] = \prod_{j=1}^{m} \rho_j^{i_j}(1-\rho_j) \tag{4.43}$$

where i_j and $\rho_j = \frac{\lambda_j}{n_j \mu_j}$ are the individual system filling and the mean local load of the interconnected queueing systems $j = [1 .. N]$ of the queueing network. If no overtaking by choosing a different path across the network is possible, then the product form property also holds for the sojourn time $\mathrm{E}[T_{f,path}]$, as shown in [136].

For *Jackson networks* we find:

- independent, possible queue filling dependent, Markovian service with rate $\mu_j(X_j(t))$,
- all arrivals from the outside are modelled by a Poisson process with rate $\lambda_{0,j}$,
- static routing matrix R with components $r_{i,j}$ that equally apply for all flows,
- no priority or other discriminatory sharing of resources,
- infinite FIFO queueing at all nodes (no losses),
- open queueing networks.

Little's law $N = \lambda T$ is applicable for entire Jackson networks if global expectations are required, as well as per node:

$$\mathrm{E}[T_{f,network}] = \frac{\sum \mathrm{E}[X_i]}{\sum \lambda_{0,i}} \qquad \mathrm{E}[X_i] = \frac{\rho_i}{1-\rho_i} \tag{4.44}$$

$$\mathrm{E}[T_{f,i}] = \frac{\mathrm{E}[X_i]}{\lambda_i} = \frac{1}{n_i \mu_i (1-\rho_i)} \tag{4.45}$$

However, note that for the global mean flow time $\mathrm{E}[T_{f,network}] \leq \sum \mathrm{E}[T_{f,i}]$ the entire system filling across all nodes $\sum \mathrm{E}[X_i]$ is divided by the ingress load from the outside $\sum \lambda_{0,i}$ only. This apparent imbalance incorporates the average path length that load units travel on their way through the network.

Finally, Jackson networks are assumed to be open. However, closed Jackson networks with finite population c also show product-form solutions, as described by the Gordon-Newell theorem [137].

BCMP networks

In [84, 1975] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios show that several more queueing systems yield *product form* results. These are in particular:

- infinite *M/M/1* systems, often applicable to model traffic sources (edge nodes),
- *M/G/∞* systems, commonly used to integrate random (transmission) delays,
- *M/G/PS* systems, which approximately model multiplexed serving (shared links),
- and preemptive work conserving (resume) *M/G/1/LIFO* systems (stacked processing).

The gained plurality of systems enables a more accurate modelling of real networks by solvable queueing networks. They are therefore called *BCMP networks*, in reference to the authors of [84] who first found and published this extension. Note, besides the restriction to the specified infinite queuing systems, the only requirement is that all traffic from the outside is Markovian. Ongoing research is focused on further extensions, on one side to find more queueing systems that can be used [63, 138], in particular how systems with losses may be integrated [139, 140], and on the other side whether the Markovian ingress restriction can be relaxed [138, 141]. Anyhow, the queueing network models can be composed of any combination of the above listed queueing systems. The distribution of network internal arrivals from neighbouring queueing systems and the merging of these is not explicitly required.

As already recognised with *Jackson networks*, the product form property provides a relation among the network state probabilities and the local mean loads present. Once we have calculated both, for example using one of the many algorithms presented in [142], we can apply Little's law $N = \lambda T$ to get the flow times per node and therefrom the performances of different paths (routes) across the queueing network.

The discussion of resource sharing systems in section 4.1 has revealed that the performance of processor sharing states an optimally fair sharing target. If we may assume that this target is sufficiently achieved, and that all queues are enough large to be modelled as infinite, then we can use a BCMP network composed solely of *M/G/PS* nodes to model the network layer of a packet switched network. In addition, we can include traffic sources that may be modelled by *M/M/1* nodes as well as random delays resulting from lower layer interconnects by $M/G/\infty$ nodes. However, differently distributed flows, as it is likely the case with MPLS for different traffic classes, can thereby not be modelled. Also the source $\rightarrow$ destination based routing, omnipresent in communication networks and particularly essential for MPLS, cannot be realistically modelled.

Multi flow extensions

The two assumptions that significantly separate queueing networks from communication networks are the probabilistic routing and the independent and identically distributed service time presumption among different traffic flows across the network. Both can at least to some extent be mitigated if we separate the traffic into *different classes*. Per class k we can then specify a fitting mean arrival and service rate per node, $\lambda_{i,k}$ and $\mu_{i,k}$, and a dedicated, only within a class probabilistic routing matrix $\underline{R}_k$. For the example depicted in figure 4.82 we assume two traffic classes $k=[a,b]$. Note that

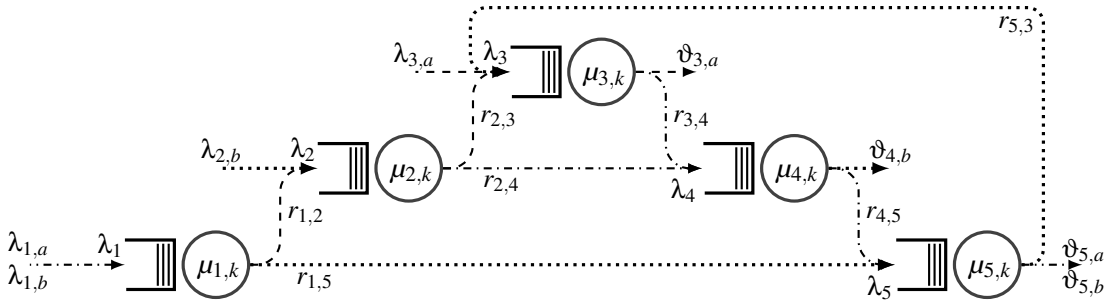


Figure 4.82: Open queueing network with class based routing

the destination of a packet arriving is not a priori given if the routing matrices $\underline{R}_{a,b}$ contain rows with more than one non-zero entry (splitting) or if any non-zero row-sum is <1, which indicate that a part of the load departs the network at the node determined by the row index. The dashed links carry solely class a traffic, the dotted links class b traffic, and the dash-dotted links carry both classes. The arrival vectors and routing matrices for the example are

$$\vec{\lambda}_{0,a}=\begin{pmatrix}\lambda_{1,a}\\0\\\lambda_{3,a}\\0\\0\end{pmatrix}\quad \underline{R}_a=\begin{bmatrix}0&1&0&0&0\\0&0&r_{2,3}&r_{2,4}&0\\0&0&0&r_{3,4}&0\\0&0&0&0&1\\0&0&0&0&0\end{bmatrix}\quad\text{and}\quad \vec{\lambda}_{0,b}=\begin{pmatrix}\lambda_{1,b}\\\lambda_{2,b}\\0\\0\\0\end{pmatrix}\quad \underline{R}_b=\begin{bmatrix}0&0&0&0&1\\0&0&0&1&0\\0&0&0&1&0\\0&0&0&0&r_{4,5}\\0&0&r_{5,3}&0&0\end{bmatrix}$$

where the nodes $3,4,5$ form a loop for traffic of class b only. A zeros-row represents an absorption node for that class because all traffic leaves to the outside. A zeros-column represents a node where traffic of that class is never routed, but may still arrive there from the outside.

In the extreme, we might define an individual class per connection and data flow to most accurately represent a communication network. The sub-networks than are always loop-free, if source and destination node are different. Obviously, the computation complexity rises with every added class and thus very detailed models are practical for rather small networks, or confined network sections only. However, to do so we need to find a method to extend the usable queueing system models to support different handling of multiple flows, each possibly having its own service time distribution.

In [83] it is shown that due to the service time insensitivity of *M/G/PS* it can quite easily be extended into *M/mG/PS* such that a multi-flow network can be modelled using the BCMP theory because the service time insensitivity renders the departure process independent of the flows the served clients belong to. Only their mean service time and mean arrival rate are relevant because the queues of the network perform as if all arrivals would be Markovian. Thus, the queue filling and flow time can be determined individually, system by system, once the mean load composition per queueing system $\vec{\lambda}$ is known.

$$\vec{\lambda} = \sum_k \vec{\lambda}_k \qquad \text{getting } \vec{\lambda}_k \text{ from solving} \qquad \vec{\lambda}_{0,k} = \underline{R}_k \vec{\lambda}_k \qquad \text{per class } k \tag{4.46}$$

Another approach is shown in [141], where the different classes' arrival streams are modelled by a decomposed Markov arrival process, called MMAP, and the analysis is based on an approximation of the departure process.

If also prioritisation has to be considered more complex approaches are required, see for example [143]. If in addition a feedback based autonomous load insertion control at every flow's source shall be considered, than a joint model considering all aspects is likely too huge and complex to allow a tractable analytic treatment. This is where simulation exhibits its strengths, and where analysis commonly retreats to coarse simplified models, for example assuming independence and optimal sharing performance. However, networks of finite systems may in principle always be modelled and solved. The problem that remains is the size of the state space, which quickly grows to astronomic regions because of many Q-matrix dimensions.

Concluding, we note that the above presented queueing network theory is not capable to consider finite queueing systems and non-Markovian arrivals from the outside. In the more recent literature we find several promising approaches tackling both. These are mathematically challenging and as we anyhow cannot precisely predict real traffic, we take next a look on simpler approximation methods based on the second moments only.

4.3.3 Queueing network approximation

To approach the problem in a step-by-step manner let us assume that the systems of the queueing network could be handled in isolation, considering the composition of the arrival and departure process only, using the available results for the individual systems. The basic assumption for this approach is *independence*, such that the random flow times $T_{f,j}$ of the systems j involved along a path across the network may simply be summed to get the end-to-end passage time $T_{f,path} = \sum_{j \in path} T_{f,j}$. In that case we can model the queueing network by a network of servers, each defined by its *sojourn time* $T_{f,j}$, now used as the *system's holding time*, as sketched in figure 4.83. This is the same network

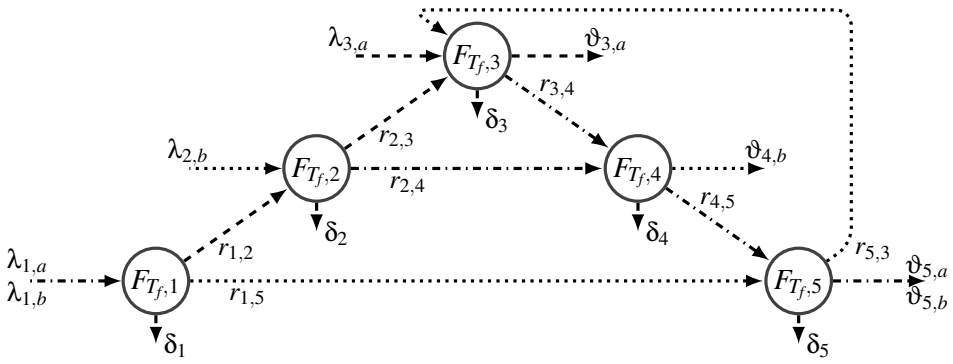


Figure 4.83: Open queueing network modelled by interconnected sojourn times

example as used before in section 4.3.2 (figure 4.82); please refer thereto for details on the assumed

network and traffic flows. In brief, there are two traffic classes $k=[1,2]$, probabilistically routed via routing matrices $\underline{R}_k$. In addition we allow now losses δ_j by integrating in the routing matrices a sink (absorbing node) that collects all lost load. Note, the loss-rates $\delta_{j,k}$ may thereby be specified per flow, allowing us to consider discriminatory queueing disciplines.

To apply the merging and splitting of flows presented in section 2.4.2 we need to assume that all $T_{f,j}$ are independent, meaning that there exists no correlation among the sojourn times of the individual queueing systems forming the queueing network. This is in general not fulfilled, but to approximately solve queueing networks by the two moments approach presented in [57] it needs to be assumed.

Recursive departure approximation – sojourn times in series

The sojourn time is the duration that a load unit needs to pass a system. It is therefore also called *flow time* T_f in accordance to the *stock-flow model* shown in section 2.4.1 figure 2.36. In terms of queueing systems it is the time that passes in between the arrival of a load unit and its departure. How this duration is composed, is on this abstraction level irrelevant. In practice, for a packet switched node, it could simply be monitored using local time-stamping and statistical evaluation.

In case of discriminatory resource sharing the sojourn time is different for different traffic classes. For the approximation we assume that discrimination works perfectly, such that the distribution of the flow time $F_{T_{f,j}}$ is the same for all traffic classes, only the means $t_{f,j}(k)$ shall be class dependent, such that

$$\mathrm{E}[T_{f,j}] = \sum_k u_{k,j}\, t_{f,j}(k) \qquad \forall_{\text{nodes } j}, \tag{4.47}$$

where $u_{k,j}$ is the mean percentage at which load form class k is present at node j, for which $\sum_k u_{k,j}=1\,\forall_j$ needs to be fulfilled at any time.

The load per system and traffic class we get from the solution to the linear system of equations defined by

$$\sum_k \vec{\lambda}_{0,k} - \vec{\delta}_k = \sum_k \underline{R}_k \left(\vec{\lambda}_k - \vec{\delta}_k\right) + \sum_k \vec{\vartheta}_k \tag{4.48}$$

constraint to

$$|\vec{\lambda}_{0,k}| = |\vec{\vartheta}_k| + |\vec{\delta}_k| \quad \forall_k \tag{4.49}$$

where $\vec{\lambda}_{0,k}$ are the class-k arrival-rates from the outside, $\vec{\delta}_k$ the loss-rates and $\vec{\vartheta}_k$ the departure-rates of class-k traffic to the outside. The constraint simply states that the load per class that enters the network from the outside also to leaves it. Having found the individual load shares per queueing system $\vec{\lambda}_k$ we can state the utilisation factors $u_{k,j}$ per class per node, being the percentages at which loads from different classes are present at node j,

$$\vec{u}_k = \frac{\vec{\lambda}_k - \vec{\delta}_k}{|\vec{\lambda}_k - \vec{\delta}_k|} \tag{4.50}$$

required to know the composition of the load leaving a node. Note, for the load shares entering a node we need to normalise the arrival vector, $\vec{a}_k = \vec{\lambda}_k/|\vec{\lambda}_k|$, and evidently, the difference $\vec{a}_k - \vec{u}_k$ states the loss shares per class.

Using the distribution merging presented in section 2.4.2 we can now node by node calculate the arrival distribution and an approximate for the departure distribution. In case of a feed-forward topology we start with nodes that face arrivals from the outside only, and move node by node through the network, recursively calculating the departure distribution of nodes once all ingress

distributions of that node are known. In case the network topology includes loops we retard to iterative re-calculation until the changes per round become negligible. Anyhow, we also calculate the mean sojourn times $\mathrm{E}[T_{f,j}]$, as these are finally required to calculate the end-to-end *propagation times* $\mathrm{E}\left[T_{f,k,path}\right]$ along paths.

$$\mathrm{E}\left[T_{f,k,path}\right] = \sum_{j \in path} \mathrm{E}\left[T_{f,k,j}\right] = \sum_{j \in path} u_{k,j} g_k \, \mathrm{E}[T_{f,j}] \tag{4.51}$$

Besides the propagation time along paths also the losses that accumulate along paths $\delta_{k,path}$ state a prime performance metric,

$$\delta_{k,path} \neq \sum_{j \in path} \delta_{k,j} \qquad \rightarrow \qquad P_{b,k,path} = \frac{\delta_{k,path}}{\lambda_{0,k}(path)} = 1 - \prod_{j \in path} \left(1 - \frac{\delta_{k,j}}{\lambda_{k,j}}\right) \tag{4.52}$$

where we cannot by default use the summation because not all class-k load present at a node j necessarily belongs to the same path. Instead, we get it via the loss probabilities $P_{b,j,k} = \frac{\delta_{k,j}}{\lambda_{k,j}}$ per class and node, precisely the product of the success probabilities $(1 - P_{b,j,k})$ over the path. Note that $\lambda_{0,k}(path)$ states the ingress rate for class-k traffic that shall be transported along a precisely identified path and that (a) the source node can be the source for many paths, and (b) different traffic classes may be transported along the same path. Here we assume that paths are defined per traffic class (flow) and that the ingress load per flow λ_k is given separately and not to be calculated from $\underline{R}_k$. Vice versa, $\underline{R}_k$ shall result from the given load volumes per flow because these state the capacity demand as it is risen by applications (the OSI-layer above).

If we can calculate the sojourn time distributions $F_{T_{f,j}}(t)$ per node, than we can as well calculate the sojourn time distributions of paths by successive convolution ($\circledast$ = multiplication of the Laplace-transforms) of the individual distributions,

$$F_{T_{f,path}}(t) = \bigcircledast_{j \in path} F_{T_{f,j}}(t) \qquad \circ\!\!-\!\!\bullet \qquad F_{T_{f,path}}(s) = \prod_{j \in path} F_{T_{f,j}}(s) \tag{4.53}$$

if we only could assume that all these distributions are independent. Actually, they usually are not. For example, the so called *streamlining effect* [56] leads to a better performance than predicted by approximation methods that disregard the existence of correlation.

Coefficient of variation approximation – aggregate composition and decomposition

Calculating all the flow time distributions $F_{T_{f,j}}(t)$ per queueing system may be exhaustive, and only considering the mean flow times $f_{f,j}$ likely yields results that diverge too much from reality. We repeatedly encountered that for many queueing systems the mean performance metrics depend on the first and second moments only. A methodical approach based on recursive coefficient of variation calculation is shown in [57], and briefly repeated here. Figure 4.84 visualises the approximation approach. First a traffic aggregate is composed that covers all ingress flows by an approximate square

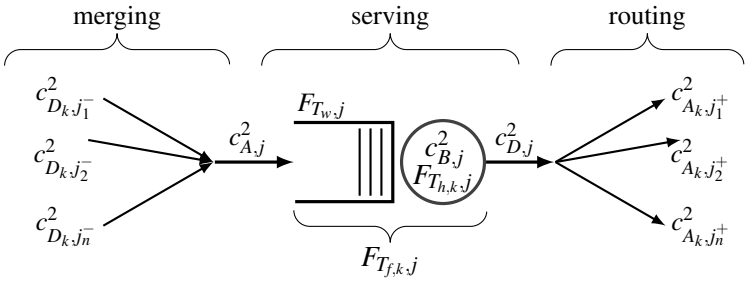


Figure 4.84: Approximation by arrival *superposition* –> aggregate *queueing* –> departure *splitting*

arrival coefficient of variation $c^2_{A,j}$. Then we assume a single queue single server queueing system and approximate its effect on the aggregate square departure coefficient of variation $c^2_{D,j}$. Finally, the departure aggregate is split into subsequent arrival components' square coefficient of variation $c^2_{A_k,j^+}$ according to the routing table $\underline{R}_k$. At downstream nodes these become superpositioned with those arriving from other upstream neighbours thereof, initiating the same procedure node by node. In the end we get the system of equations stated in equation 4.57.

Note that in our perspective, in respect to communication networks, the queueing systems constituting the queueing network represent the links of the communication network, and not the nodes thereof. Thus, the topology of the queueing network is the dual topology of the communication network: nodes become links, links become nodes. For star and ring topologies the two are structurally identical, in general the topologies differ. However, due to that, the *superposition* refers to the node internal merging of flows at egress ports, and the *splitting* refers to the routing of the ingress load composites from all ingress ports to different egress ports.

Assuming renewal property for all processes an equation to calculate the squared arrival coefficient of variation $c^2_{A,j}$ is derived in [57] based on [58]

$$c^2_{A,j} = \frac{\lambda_{0,j}}{\lambda_j} c^2_{A_0,j} + \sum_k \frac{\lambda_k u_{k,j^-}}{\lambda_j} c^2_{D_k,j^-}, \tag{4.54}$$

where $c^2_{D_k,j^-}$ is the squared departure coefficient of variation of flow k when leaving the preceding resource j^-, and $c^2_{A_0,j}$ is the squared arrival coefficient of variation of all arrivals entering the network at resource j. Thus, the first term refers to the ingress flow entering the network at resource j, providing the given part of the system of equations that relate these to the second term contributing the a priori unknown arrivals from neighbouring nodes j^-, where the traffic leaves as departures.

Assuming that the per flow departure characteristic is related to the aggregate characteristic only, the approach outlined in [57] continues by using the splitting proposed under the assumption that the departure process is of renewal type, to get an equation for the squared departure coefficient of variation per arriving flow k from node j^-.

$$c^2_{D_k,j^-} = u_{k,j^-} c^2_{D,j^-} + (1 - u_{k,j^-}) = 1 + u_{k,j^-}\left(c^2_{D,j^-} - 1\right) \tag{4.55}$$

For the aggregate's departure coefficient of variation c^2_D of a node, a simple and rather common approximation is used

$$c^2_D = \rho^2 c^2_B + (1-\rho^2) c^2_A = c^2_A + \rho^2\left(c^2_B - c^2_A\right) \tag{4.56}$$

where $\rho = u_j$ is the utilization of the resource j, and c^2_B is its aggregate service time coefficient of variation. It depends on the aggregate arrival characteristic c^2_A entering the resource j, which needs to be calculated likewise based on the flows entering it. This proceeds across the entire network of resources, defining a system of equations that is linear in terms of squared coefficients of arrival and service variations c^2_A and c^2_B

$$c^2_{A,j} = \frac{\lambda_{0,j}}{\lambda_j} c^2_{A_0,j} + \sum_k \frac{\lambda_k}{\lambda_j}\left(u_{k,j^-} + u^2_{k,j^-}\left(c^2_{A,j^-} - 1 + \rho^2_{j^-}(c^2_{B,j^-} - c^2_{A,j^-})\right)\right) \qquad \forall_j \tag{4.57}$$

that can be solved rather easily, once we solved 2.101 to get the load shares $u_{k,j}$ at each resource j for every flow k present. To do so we need given ingress flow intensities λ_k and ingress coefficient of variations $c_{A_0,k}$, all flow's routing across resources, being node j sequences per flow k, and the service time coefficient of variation c^2_B per resource, which may be composed from the present flows according to 2.100, adding one more dimension.

In general equations 4.54, 4.55 and 4.56 are approximations only, because all contributing flows have to be independent and of renewal type to make this approach precise. Independence of ingress flows is not necessarily preserved, for example if two flows pass a common preceding resource their

arrivals to downstream nodes may be correlated. The renewal property can be assured only if all initial flows result from Poisson processes and all service times across all resources are negative exponentially distributed. This is the case for *Jackson networks* [133] but not in general.

Finally note that in general the results calculated for a traffic aggregate cannot be decomposed into performance metrics per contributing flow. To evaluate systems accurately on a per flow level, it is possible to use *vacation models*, where the serving process becomes split among different traffic flows. Per flow a sub-model is designed, where the serving is interrupted to model the service of load units from other flows. This is a contrary approach: the individual service time distribution is preserved, which is possible because without preemption only the waiting time prior service depends on the properties of the other flows currently present. However, the effects of a shared finite queue can thereby not be covered. A variety of vacation models is for example discussed in [60] and in most comprehensive text books on queueing systems.

Departure process approximation – the *GI/G/1* queue

In section 3.1.5 we defined $T_X(n)$ to be the time lag between service completion of client n and service start of client $n+1$. Thereby the inter-departure time may be stated as

$$T_D(n) = T_X(n) + T_S(n+1).$$

Due to independence of T_X and T_S we get for the first two moments

$$\mathrm{E}[T_D] = \frac{1-\rho}{\lambda} + \frac{1}{\mu} = \frac{1}{\lambda}$$
$$Var(T_D) = Var(T_X) + Var(T_S)$$

where the result for $\mathrm{E}[T_D]$ is evident for any infinite queueing system. Applying the calculation rules for variance and covariance, the equation found for $Var(T_D)$ lets us express the departure variance by first and second moments of the arrival and service distribution and the mean waiting time[3].

$$Var(T_D) = \sigma_A^2 + 2\sigma_B^2 - 2\frac{1-\rho}{\lambda}\,\mathrm{E}[T_W] \tag{4.58}$$

This is a nice finding: it tells us that once we know the mean waiting time we also know the first two moments of the departure process, and thus the two moments required to at least approximately characterise the outgoing flow in case we want to analyse a cascade of queuing system. In particular, with flow based communication networks we may simply monitor the mean flow time and thereby reveal locally some information upon the departing flow.

Inserting the bounds found for the waiting time in section 3.1.5 equation 3.52 we get upper and lower bounds for the departure variance.

$$\sigma_B^2 \leq Var(T_D) \leq \sigma_A^2 + \sigma_B^2 + \frac{1-\rho}{\lambda^2\mu} \tag{4.59}$$

On one side the departure process cannot be less variant than the service process, and on the other side, the upper bound depends on the current load. Likely the actual variance of the output process does so as well. Stating an output distribution that identifies the departure process based on the variances of the contributing processes only, independent of the current load, is rarely possible. Figure 4.85 shows the dependence of the inter-departure time coefficient of variation

$$c_D = \frac{\sqrt{Var(T_D)}}{\mathrm{E}[T_D]} = \lambda\sigma_D$$

on the load ρ. Only for $c_S = c_A$ is c_D load independent.

[3] $Var\left(T_W^{(n+1)} - T_X^{(n)}\right) = Var\left(T_W^{(n)}\right) + Var(T_S) + Var(T_A) \rightarrow Var(T_S) + Var(T_A) = Var(T_X) + 2\,\mathrm{E}[T_W]\,\mathrm{E}[T_X] \rightarrow Var(T_X)$

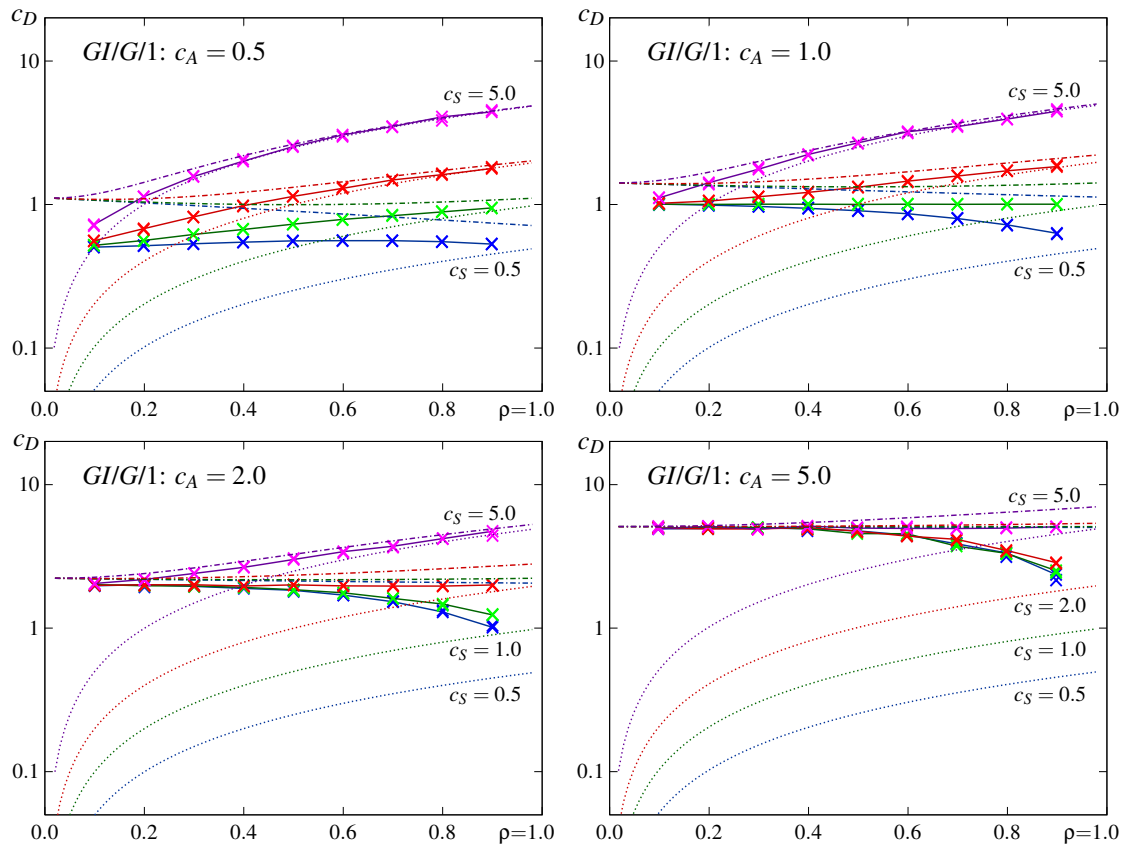


Figure 4.85: Departure coefficient of variation c_D for $GI/G/1$ queueing systems with $c_{A|B}$=[0.5, 1, 2, 5] respectively, upper bounds (- · -), lower bounds (· · ·), simulation results (×)

The lower (dotted) and upper (dash-dotted) bounds depend not only on the load $\rho = \frac{\lambda}{\mu}$ but also on the actual rates, λ and μ. Therefore, we show in figure 4.85 the upper bounds calculated for $\lambda = [0\,..\,1]$, $\mu = 1$, and the lower bounds for $\lambda = 1$, $\mu = [\infty\,..\,1]$, to get the general bounds applicable for all λ/μ-ratios yielding the same load ρ. Also the simulation has been performed for both cases, and therefore there are actually two overlayed simulation results × shown. However, except from statistical fluctuation the simulation yields the same results for both cases.

Note that the lower bound is defined independent of the arrival process. Consequently, it is the same in all sub-figures. That it approaches zero at zero load is evident because it needs to hold as well for deterministic arrivals with $c_A = 0$. As commonly the case with infinite queueing systems, the impact of a bursty distribution can hardly be compensated by a smooth other process. Therefore, in case of smooth arrivals and bursty service the departure variance rises quickly, and vice versa, for bursty arrivals and smooth service it remains at a high level until more than 50% load is reached. If the variance of the arrival and the service process are the same, we get a more or less horizontal line. For *M/M/1* it actually is, according to theory, a straight line. For the smooth-smooth combination it shows a marginal bend upwards. At the extremes, $\rho = 0$ or 1, c_D approaches the arrival and service variance, c_A or c_S, as could be anticipated.

This non-linear dependence of the output variance on the system load challenges the calculation of cascaded systems; particularly, if the load is unsteadily varying. An extreme example is pure on/off-traffic, as it can be observed on access networks, comparable to busy streets controlled by traffic lights: green = get some content, red = use received content. Such systems should be modelled for selected load levels (for example *high* and *low*) because a *mean* distribution that results for the average load $\rho = \frac{t_{on}}{t_{on}+t_{off}}\,\rho_{on}$ will commonly not compare with the perceived performance [130].

4.3.4 Quality of service

In the introduction to this chapter, figure 4.1, we recognised the different traffic management approaches and their scope. All these cooperate in order to deliver the *quality of service* QoS required to provide a data transmission based IT-service. These services are very different in their demands, as listen in table 4.4 [130]. For the traditional data-centric applications the jitter is irrelevant because

Table 4.4: QoS requirements of different IT-services (applications)

application	delay	jitter	loss probability
voice (telephony)	150 ms → good 400 ms → least	30 ms	1 %
interactive video, video-conferencing	100 .. 400 ms	50 ms	0.1 %
video broadcast, video streaming	4 .. 5 s	—	< 5 %
interactive IT services, messaging, chat	150 .. 400 ms	—	1 .. 5 %
downloads, web-pages, e-mail	unspecified	—	1 .. 5 %

they were designed to operate with unreliable data transmission, foreseeing an inevitable buffer at the destination to collect the correctly received packets until the entire content can be reconstruct, independent of the timing and order of packet arrivals. In these cases, the delay constraint mainly refers to the service's response time to user interaction, shown in table 4.7, not to be mistaken with the sojourn time.

In contrast to traditional unicast IT-services raise interactive services, where end systems communicate seemingly in real time, not only stringent delay constraints. The jitter constraints of traditional telephony services is basically risen by the signal processing performed at end systems, in particular if these were designed for jitter free circuit switched lines. Even more restrictive is the loss constraint because for many so called '*real time*' applications lost packets are a serious problem: they trigger re-transmissions, which increases the delay and causes massive jitter, leading to short but annoying service interruptions.

Continuous services alike multimedia streaming should therefore avoid any retransmissions and encapsulate sufficient redundancy in the content compression and encoding (*error correction bits*) to mitigate the quality degradation when skipping lost segments in the reproduction of the content, utilising for example *video frame buffers*. In addition, the compression should be sensitively adjusted to the transmission quality (QoT) in order to achieve maximally interruption free content delivery.

A completely different story is synchronisation, in particular lip-synchronism. Because packet switched networks are by design asynchronous, synchronism either needs to be enforces by the encoder, packing video and audio information in parallel into the same transport containers, for example using the audio channels provided by SDI frames, or by the playback device. Latter demands identical time-codes encapsulated in the first place in both signals, audio and video. However, any off-line synchronisation as well as heavy compression cause processing effort prior final delivery, causing additional latencies.

On the other side demand these services very different transport capacities, shown in table 4.5 [130]. From this table it becomes very clear that the major capacity drivers of today are the video services. Recent rumours tell that more than 50 % of today's Internet traffic is caused by video services. Considering the high quality demand, it becomes evident that today the Internet has to deliver a QoS unparalleled to what it has been designed for. Thus, all mechanism available to improve the end-to-end QoS need to be maximally exploited. Still, the good performance we experience in developed countries is mostly gained from massive over-provisioning. Premium network operators run their core networks at an average utilization far below 30 %, and they increase the capacity, for example by adding/activating more line-cards, whenever the peak load reaches the 50 % margin.

Table 4.5: Transport capacity demand of different IT-services (applications)

application	capacity demand
tele-text	0.3 kbit/s
web-pages, file-transfers, e-mails, etc.	as much as it gets
digital music streams (web-radio, HD-audio)	32 .. 4800 kbit/s
video conferencing (per attendee)	0.384 .. 2 Mbit/s
MPEG-2 video streams (DVD, satellite)	2 .. 8 Mbit/s
MPEG-4 video streams (SDTV, HDTV)	1.5 .. 20 Mbit/s
uncompressed digital video (SDI)	0.3 .. 3 Gbit/s
digital cinema (2k, 4k, iMAX)	1.5 .. 20 Gbit/s

Over-provisioning is for sure the most effective approach, fundamentally supported by all performance studies presented in chapters 3 and 4. However, to maximise the efficiency, network engineers always seek for an optimal configuration of the many systems involved in the transmission of information that achieves the best QoS possible form the available/added resources, including capital expenditures and operational costs. Keeping an eye on economics and customer satisfaction, it is clear that the efforts put in QoS maximisation will not exceed the gained returns and may not spoil any service. Actually, QoS degradation in the context of packet switched networks is primarily caused by load peaks, which becomes equally aware from all the performance studies presented in chapters 3 and 4. Thus, the most efficient approach to QoS maximisation within packet switched networks is the avoidance of load peaks.

Quality of Experience – the end-to-end view

Quality of Experience (QoE) assesses the customer's satisfaction with a service. It focuses on a vendor's offering solely from the standpoint of the service customer. In this context we first need to extend our view. The true chain of systems relevant for the perceived performance, sketched in figure 4.86, includes not only the data transport, it also includes customer equipment out of the control hemisphere of the service provider. Most delays caused by end systems are easy to

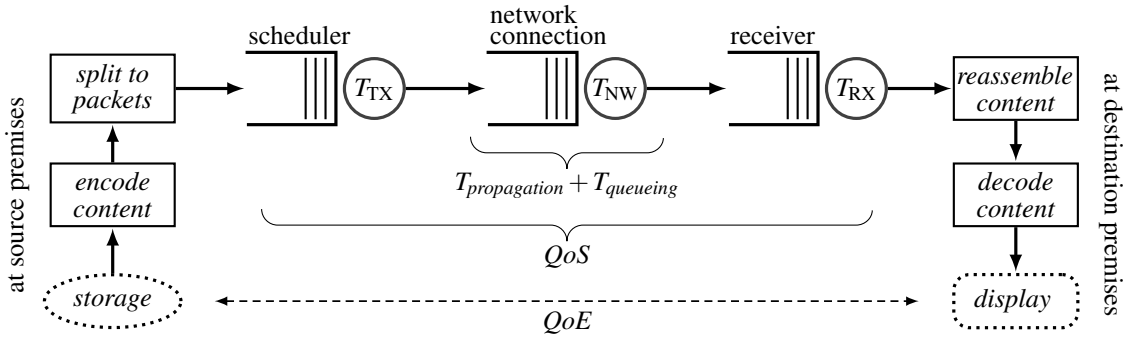


Figure 4.86: Complete end-to-end path and scopes of QoS and QoE metrics

asses from equipment tests. However, variable delays as for example caused by the decompression of variably compressed signals, need to be considered by their worst-case figures. The other uncontrollable component is the inevitable propagation delay, which results from the finite speed of signal propagation in different physical media. Table 4.6 lists exemplary figures for the propagation delay of typical communication links to gather the ranges where these are relevant. Wired links across free space are rather impossible and therefore not listed. For a technology that foresees for example 10 ms clocking at every node, are propagation delays of distances up to hundred kilometres rather negligible. On the other side, the propagation delay may also prohibit some services. For

Table 4.6: Approximate signal propagation times of common transmission media

distance	free space	copper cable	silicon fibre
1 km	3.34 μs	4.45 μs	5.34 μs
300 km	1.00 ms	1.33 ms	1.60 ms
5.000 km (transatlantic)	16.7 ms	22.2 ms	26.7 ms
44.000 km (around the world)	147 ms	196 ms	235 ms
72.000 km (geostationary satellite)	240 ms	—	—
1 au= $149.6 * 10^6$ km (inter-planetary link)	499 s	—	—

example is video-conferencing with attendees distributed around the world difficult due to the limited achievable performance, independent of the provided connection capacity.

Next we specify *response time ranges* based on personal observation and our interpretation of rather vague recommendations found for different applications. Note, the ranges stated in the right table 4.7 lack reference and approval because they are intrinsically subjective. The usability of

Table 4.7: User perception of system response latencies and MOS factors [144, 145]

response latency	perceived as
$<$ 10 ms	*seamless*
10 .. 100 ms	*instantaneous*
0.1 .. 1 s	*acceptable*
1 .. 10 s	*heavy delayed*
$>$ 10 s	*unresponsive*

MOS factor	MOS rating	degradation is
5	*excellent*	imperceptible
4	*good*	perceptible
3	*fair*	slightly annoying
2	*poor*	annoying
1	*bad*	very annoying

the here stated quality ranges depend on the provided service. However, they always respect all subsystems involved according to figure 4.86: from the source of the signal remotely controlled by the customer, till the end system at customer premises providing the interface to a human user. For other user types, for example machine-to-machine communication, other, in a wide range tunable figures need to be considered. Human users are rarely tunable and may respond irrational if manipulated. Better not to be tampered.

The mean opinion score (MOS) shown on the right in table 4.7 refers to a test commonly used with telephony networks to obtain the customer view of the perceived quality. It is based on averaged subjective quality ratings by a comprehensive selection of test customers, individually scoring the service quality as they perceived it in a 'quiet' test environment according to ITU-T recommendation P.800 [145]. For voice over IP (VoIP) ITU-T PESQ P.862 [146] states a method to calculate MOS factors directly from IP network performance. However, this method skips many subsystems of the processing chain. In consequence, the different reaction of these sub-systems on the loss or delay of different parts of the content is missed. Thus, these calculated MOS factors can severely diverge from the MOS ratings achieved using the original method defined in ITU-T P.800.

For multimedia services the MOS provides a numerical indication of the perceived quality from the users' perspective on received media services, including all mechanisms of the content processing and transportation chain shown in figure 4.86. In addition, several aspects influence the quality rating, not only the perceptibility. See for example the QoS categories introduced in ITU-T recommendation G.1010 [144]. A clear drawback of MOS is the demand for a representative group of test persons. This may be expensive, but more importantly it is very time consuming and difficult to implement in parallel to system development and deployment.

Concluding, the end-to-end service quality is difficult to asses and depends on the sensibility of the subsystems involved on the transmission quality. Thus, QoS specifies the performance of a connection across some network, and QoE the performance of an IT-service. The relation among QoS and QoE depends on the subsystems required outside the network to deliver the service.

Performance expectation – how to satisfy customers

Regular consumers of a service are commonly aware of the technically inevitable shortcomings and accept these. Thus, the customers of queueing systems and chains thereof are primarily displeased by the varying waiting time T_w because this part of the flow time T_f does not always occur and seemingly comprises no necessity.

For the operators of packet switched networks the waiting times T_w at network nodes state the price to be paid for an economically satisfying utilisation of the provided resources and is *intrinsic to statistical multiplexing*. Thus, the target of network optimisation is to parametrise the involved queueing systems j such that the maximal accumulated mean waiting time $\mathrm{E}[T_{w,k}] = \sum_j \mathrm{E}[T_{w,j}]$ along paths k across the network is minimal and at the same time maximally constant, meaning that the time derivatives $\dot{T}_{w,k}$ called *jitter* shall be minimal as well.

$$\min_k \left(\mathrm{E}[T_w^{[k]}] \right) \quad \text{AND} \quad \min_k \left(\dot{T}_w^{[k]} \right) \qquad \forall_{\text{paths } k}$$

How to achieve this target in an effective and efficient manner, is an open research topic that can be addressed from different sides.

- **global:** To achieve the best possible performance the load of all network nodes shall be minimised. This is achieved by *load balancing*. For a reservation based end-to-end (source-) routed network management this is achieved by a load sensitive routing algorithm. For example based on the resource reservation protocol for traffic engineering (RSVP-TE), which is commonly used with MPLS. However, using longer paths than the shortest possible always increases the total network load because longer paths occupy more resources. For plain IP networks this approach is impossible because there exists no reservation of resources. Such can be integrated by add on protocols, for example MPLS, which add their own resource management turning the network from connection-less into connection-oriented.
- **sectional:** With hop-by-hop routed networks a timely accurate view of the current network state is hardly achievable, and anyhow excessive for thousands of nodes. For these *deflection routing* can divert some load to alternative paths from a currently overloaded node if this information is timely propagated to upstream nodes. However, path toggling should be avoided. It may be necessary to re-route other flows/paths to free the resources required for flows/paths that cannot be re-routed else. This demands so called autonomous systems (AS), representing a manageable part of the entire network, using so called *path computation elements* (PCEs) located at one node of the AS. PCEs combine the information from within the AS with that exchanged among neighbouring ASs and can thereby decide to either re-route a flow within their section or deflect it to an AS that advertised sufficient capacity.
- **local:** On a local scope the total load present at a node cannot be changed. Thus, the node can only try to serve the load shares in a way that fits different demands best. With MPLS the so called *per hop behaviours* are enforced. These can for example be achieved by *weighted fair queueing* (WFQ) presented in section 4.1.3 in combination with *weighted random early detection* (WRED) presented in section 4.2.2. However, as exemplified with the evaluation of these discrimination schemes, any improvement for a flow compels degraded performance of other flows. Still, this is today the most common approach. In combination with demand advertisement, as it is for example communicated by RSVP-TE, the burden on less privileged flows can be controlled. Without load advertisement any local approach is a rather blind and unreliable attempt toward QoS provisioning.
- **flow centric:** A more advanced approach toward on demand privileging is the *time-to-live* based prioritisation of arriving load units. Per hop the time-to-live is reduced by a constant or an amount that reflects the time spent at the node. Privileging load units with a shorter time-to-live shall assure that all load units reach their destination in time. This is particularly important for continuous flows that transport for example a multimedia signal. Delayed delivery is for these forbidden. Load units with expired time-to-live can be dropped without consequences.

The time-to-live header parameter can for example be used as weighting factor with WFQ. Actually, time-to-live based scheduling also *reduces the resource waist on dropped packets* because loads closer to the destination are privileged and less likely dropped than those that have not travelled a long distance already. However, today the time-to-live is widely used to remove stray load that somehow lost its path and wanders around the network. Subtracting the time spent per packet also causes processing effort and is thus not very economic.

- **feedback based:** A flow consists in general of many subsequently transported packets that traverse the network from the same source to the same destination. If the nodes somehow learn about the end-to-end performance per flow, and from time to time may try alternative paths to see if these are better, then *self-learning* could maintain the privileging and routing tables. However, packet acknowledgements do not necessarily propagate along the reversed forward path and rarely contain a quality feedback. Thus, to tell the nodes about the final end-to-end performance of a path this information needs to be propagated among nodes via signalling messages, for example re-using the *label distribution protocol* (LDP).
- **swarm controlled:** Alternatively, nodes could justify their routing tables and weighting factors solely based on those of their neighbouring nodes. If a terminal node is not satisfied with the performance a flow achieved, it privileges that flow higher and thereby triggers its neighbour nodes to do so as well. Neighbour by neighbour the change is communicated such that in the end the flow gets better privileges from its source till the destination. Evidently, nodes shall not blindly copy any changes. They need to adjust these to their current situation. Thereby, the change's magnitude is reduced neighbour by neighbour, individually considering the capabilities of each node. In consequence also the routing may change, if for example a neighbour can adjust better than a previously used node. Also heterogeneous network architectures are no problem, every node performs as it can. Finally, a terminal should reduce the weighting if the performance is better than necessary. This avoids run-away privileging.

The problem of network optimisation in terms of QoS provisioning is extended by one more dimension because most carrier networks are composed of several network layers: typically a circuit switched layer providing constant but potentially adjustable line rates (e.g., WDM, OTN, SDH/Sonet), a virtually circuit switched layer providing resource reservation options and the utilisation of reserved but not used resources for unpretentious flows (e.g., ATM, CGE), and the packet switched layer (e.g., IP) to interconnect the connections provided by the lower network layer. Actually, IP has no physical layer and therefore depends on the existence of an underlying technology to complete it by providing the necessary transmission capabilities.

If a packet switch is overloaded a viable solution to resolve the situation is to change the underlying virtual topology provided by the next lower network layer such that some load gets bypassed. Note, OTH, SDH/Sonet, Ethernet, ATM and siblings thereof, all have their own network layers besides the lower layers that provide signal transmission. The IP-centric OSI models commonly found for example in *wikipedia* are too simplified to be useful. The hierarchy defined with *generalised multi protocol label switching* (GMPLS) addresses this issue and many options to perform *multi-layer routing and resource assignment* have evolved and been published since its introduction. A very interesting approach is for example presented in [147] based on convex optimisation.

The quality of service issue is best approached by an optimal balance between queue length thresholds, to limit the worst case delay, and weighting factors, to achieve the required throughput per flow. However, common queueing network theory is based on infinite queues, whereas good QoS is achieved with minimal queues.

4.4 Future network operation schemes

In recent years the approach to network organisation and maintenance changed several times. The traditional bottom-up provisioning specified by the OSI model for digital communication systems in [1] is depicted in figure 4.87. Note, layer 1 is here renamed to *bit-layer* because the

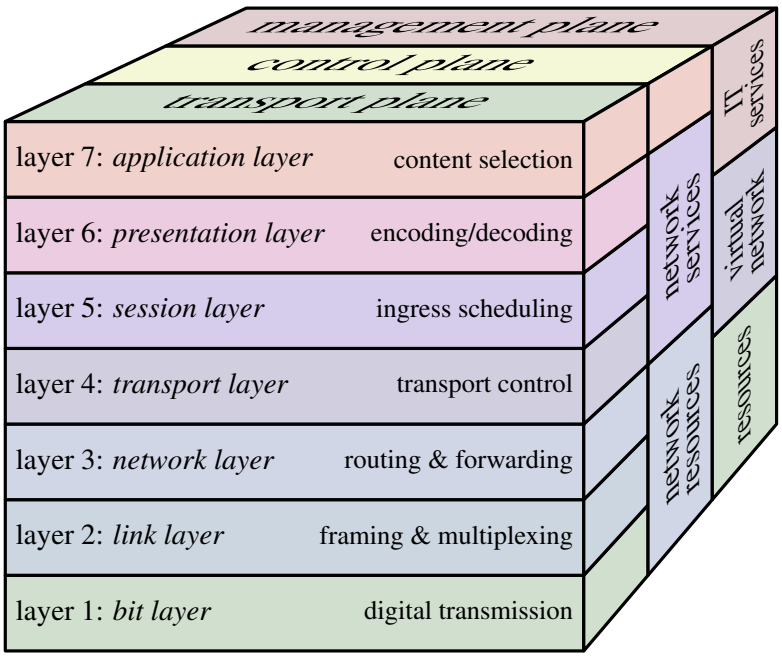


Figure 4.87: The OSI reference model, ITU-T X.200, extended by control and management planes

original name *physical layer* is misleading: all layers of the X.200 reference model refere to *digital comminication*, and that does not include physical resources alike bandwidths around carrier frequencies or wavelengths, nor the modulation formats used to convert bits into analogous symbols that are able to propagate along a physical medium. The layer 1 represents solely the *binary channel* that can transport a certain number of bits per second, which yields the *line-rate*. Therefore, the layer 1 performs digital processing toward bit-error correction in order to compensate the impairments of the physical signal propagation, but it does not per se specify the physical transmission, even though it commonly is very specific to the physical medium used for the analogous physical transmission.

The extension by a *control* and a *management plane* is done in respect to the concepts presented shortly in this section. The split in the control plane refers to the different locations: distributed in the network or concentrated in a terminal. In the management plane the network layer is partly in the scope of the resource management and partly in that of the virtual network in order to integrate multiple network layers, explained in section 4.4.2.

Widely the basic model, meaning the strict transport layers, are identified as an obstacle, hindering the development of efficient operation schemes that integrate multiple transport and transmission layers. However, the X.200 model still defines the principal functionalities, and most technologies orient their features on this model. In its principles this hierarchy has to be served, only the layering can be modified such that the entire functions-stack becomes more flexible.

The currently favoured approach, the *software defined networks* (SDN) paradigm [148–150], is intended to hide the seven layers of the OSI model by three organisational planes: a *resources plane*, a *control plane*, and a *communication plane*. Latter provides an open plethora of message parsers required to uniformly communicate with all the different network components that exist and may show up in the future. Today, the *OpenFlow* software is assumed to best serve SDN implementations [151,

152]. For the physical resources' management we could use GMPLS [153]. Possibly not the best candidate because GMPLS implements a strict packet over circuit over wavelength hierarchy, re-integrating the restrictions novel network paradigms wish to overcome.

In [13] we analysed several options by evaluating how GMPLS fits the demands of *optical burst switched* (OBS) networks. Basically, it does not. We therefore favour a flow centric approach, called *flow transfer mode* (FTM) [10], over GMPLS and clearly specified the functionalities required to become a multi-purpose network paradigm. Sloppily, FTM may be seen as generalised ATM, enabling huge variably sized physical layer cells, called bursts, that can be transmitted over fibre, coper, and radio likewise and interchangeably. A clear drawback of any burst switching technology is the bursty transmission. Mostly because of the time it takes to fill-up a reasonably sized burst. Legacy *transport control protocols*, designed for the transmission of packets over a sequence of switched circuits, cannot handle the highly fluctuating *round trip time* (RTT) caused by burst transmission. The RTT of packets should be replaced by the RTT of bursts, such that the burst transmission is controlled by TCP, not the transmission of encapsulated packets. But that is a revolutionary approach challenging the entire TCP/IP protocol suite, likely not integrable in a smooth migration path. Considering the IPv6 success, such an approach hardly can succeed.

In the following we leave the popular path and define three functional blocks that in our opinion best serve to allocate the restrictions, services, functionalities, and features provided by communication networks. Later on we sketch a lean approach to network management and very briefly how autonomous network sections (ASs) may be able to achieve this.

4.4.1 Physical network: resource management

The base of any communication network are the network components that provide the transmission capability. Without these no network can exist. We refer to them as the *resources* when referring to the features they offer to transport information encapsulated in some formatted date stream. Physically, they offer a bandwidth, which is either a frequency band, a time-slot, a combination thereof, or more abstractly a mean bit-rate. Note, a data packet has no physical meaning: it has to be converted to a physical symbols-stream/-pattern before it can be physically processed.

In general the nature of the physical resources and symbols is not specific to the data transmission. Only the limitations and impairments of the transmission and processing of the analogous physical symbols needs to be considered. These are expressed rather abstractly by three basic metrics,

$$\begin{aligned} C(j) &\ldots\ldots \text{ capacity (line-rate) } [bit/s, bps] \\ t_{prop}(j) &\ldots\ldots \text{ propagation delay } [\mu s/km] \\ P_{err}^{[bit]}(j) &\ldots\ldots \text{ bit error rate (BER) } [1] \end{aligned}$$

which define the quality of every physical resource j in terms of its data transmission capability. A such specified resource we call *channel*, irrespective of how the transmission capability is physically accomplished. In terms of resource management, the channel is the smallest assignable unit.

Physical resources can offer different and sometimes even adjustable channels. This is commonly achieved by different *multiplexing* schemes that split the total available capacity into convenient shares. Also the opposite is possible, resources may as well be bundled in order to offer channel capacities that a single resource cannot provide. The configuration of resources in order to offer different channels is the first resource management aspect.

The other resource management aspect is the provisioning of chained sequences of compatible channels, providing a channel that spans across several resources. This is commonly referred to as *switching* because it is similar to connecting wires by a physical switch. A clear drawback of the channels concept and *circuit switching* in particular, is the fact that the configured channels provide

a defined capacity irrespective whether it is used or not. Therefore, the channel capacities should in theory be minimised in order to maximise the flexibility. This lead to the tiny cell-size of 48*byte* plus 5*byte* headers used with ATM, which caused a header processing effort that did not scale well to support high capacity connections in the $Gbit/s$ range.

Today the trend fosters huge channels that are a posteriori used in parallel by software defined, auto-adjusting sub-channels. MPLS with different per-hop behaviours for different encapsulated flows is an example thereof. However, this option is not part of the physical resource management, it belongs to the virtual network's capacity management addressed in the next functional block.

An alternative to demand driven channel bundling (or splitting) is to use the available channels only for demand defined time periods. This concept is commonly referred to as *burst switching*. Here a resources capacity becomes assigned only for the period of time it is actually used. This option implements *statistical multiplexing* of physical resources. This is in particular appealing if the channels offer such huge capacities that a single data flow commonly cannot utilise it efficiently. A native candidate are optical channels that offer capacities in the range of several $Gbit/s$.

Optical burst switching (OBS) as physical layer switching paradigm is appealing because of its simplicity. However, studies comparing packet over OBS with packet over *optical circuit switching* (OCS) show that for many scenarios OCS outperforms OBS [42]. A critical issue of OBS is the amount of resources wasted in case a burst does not reach its destination, which for a queue-less technology is quite likely if the load becomes too high. Many approaches to mitigate this problem have been published. Technically excellent, they commonly cannot compete with OCS in terms of quality of transmission and achievable resource utilisation. OBS demands huge numbers of parallel available optical channels to become competitive. The solution thereto, as specified in [13], is the support of the different connection types sketched in table 4.8. The signalling options

Table 4.8: FTM channel/service types and signaling options

channel/service type	one-way	two-way
continuous wavelength (λ service)	hardly an option	strongly advised
constant bit-rate (line service)	possible	advised
adjustable bit-rate (dynamic line)	possible	advised
assured variable bit-rate (assured mean)	possible	advised
available bit-rate (virtual channel)	best choice	limited
assured single burst (possibly huge)	hardly an option	strongly advised
plain single burst/packet (rather small)	best choice	limited

stated highlight that the commonly assumed *just in time* (JET) one-way signalling proposed for OBS in [41] is not sufficient in practice. Two-way signaling is always an option, and some service types recommend it. Services that grant assured delivery without loss monitoring and re-scheduling of lost bursts demand the use of an acknowledged reservation scheme, and also those that grant a certain channel capacity or enable capacity adjustments profit from acknowledged reservation. A relational prioritisation, as outlined in [154, 155], can still be applied among the services using one-way reservation, and thus, the relative service classes defined for IP are fully supported. Using table 4.8 we can derive the resource reservation options (*signalling message*) required:

- infinite duration reservation and dedicated tear-down messages (*wire lines*),
- requesting the repetitive switching of identical bursts (*time-slots*),
- connection capacity adjustments (*adjustable lines*),
- explicitly routed single burst advertisement (*timed bursts*),
- single burst advertisement with acknowledgement (*bursts to be polled*),
- JET one-way signaling for best effort traffic (*post-and-forget bursts*).

Not explicitly mentioned are the different acknowledgement demands and routing options. Especially for routing in combination with two-way signaling either well known schemes need to be adopted or new developed in order to maximize the likelihood of routing success.

The design of an FTM switch, as sketched in figure 4.88, is identical to that of a generic OBS

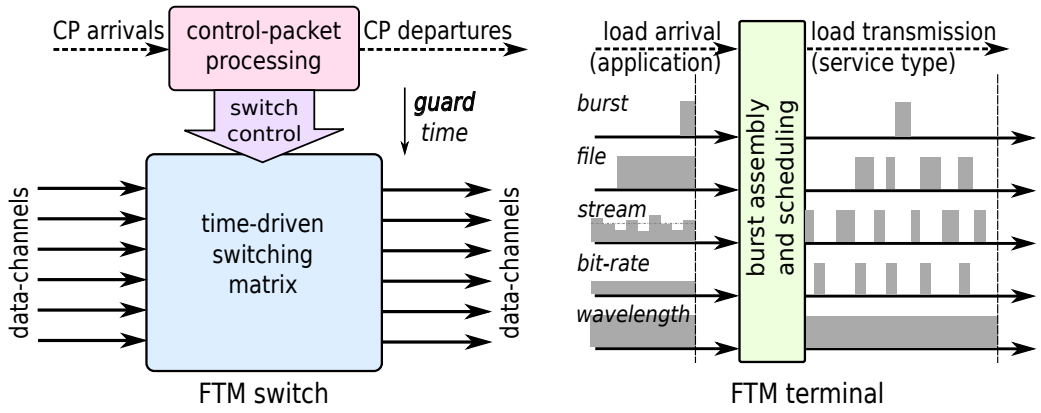


Figure 4.88: Functional blocks (a) and services (b) of an FTM-node

switch. However, FTM adds the option to realize it in the electrical domain and therefore allows to consider buffering options and rate-conversion options within the line cards of FTM nodes.

Basically, the connection types listed in table 4.8 are required for any physical resource sharing technology that is capable to adjust established channels. However, as with IP, the most demanding feature for any physical network plane is its exchangeability and interconnectability. Due to the vastly different channel capacities of different technologies this is difficult to achieve. To cover this the OSI model specifies the link layer (layer 2), where the capacities of the lower layer are structured into timely segments that can be used individually. Basically, this specifies the interface to the network layer where end-to-end routing is performed. But it also provides the features required to design virtual networks, where the channel chains provided end-to-end, called *connections*, are dynamically configured in a way that supports the demands of the flows they serve.

4.4.2 Virtual network: consistent capacity management

Primarily, the virtual network serves to join the resources provided by different technologies and domains (operators) via so called *gateway nodes*. Real networks consist of many domains that are independently realised with a variety of technologies. Each domain may be split or combined into *administrative sections*, called autonomous systems (ASs) [RFC1930]. Paths across ASs are achieved using the *border gateway protocol* (BGP) [RFC4271]. Note, ASs represent organisational sub-networks of the entire virtual network. The sub-networks may overlap considerably, meaning that nodes may be part of more then one AS. These multi-homed nodes are commonly used as border gateway nodes (bg_{xyz}), as shown in figure 4.89. Paths across AS borders commonly pass through these border gateway nodes. Which AS shall be used to bridge the gap between two AS that are not directly connected by a border gateway node can be configured by weights and policies. In figure 4.89 AS1 to AS3 may for example represent international networks, AS4 and AS5 the national networks of competing network operators, and AS7 to AS8 the access networks of different service providers.

The exchange of information among ASs is minimised and independent of the methods and technologies used within ASs. Thereby the heterogeneous networks, including wired and wireless technologies, have been enabled. For the date packets transmitted, this is an easy task. In the simplest case they only change the envelope consisting of packet/frame header and tail, in the worst case

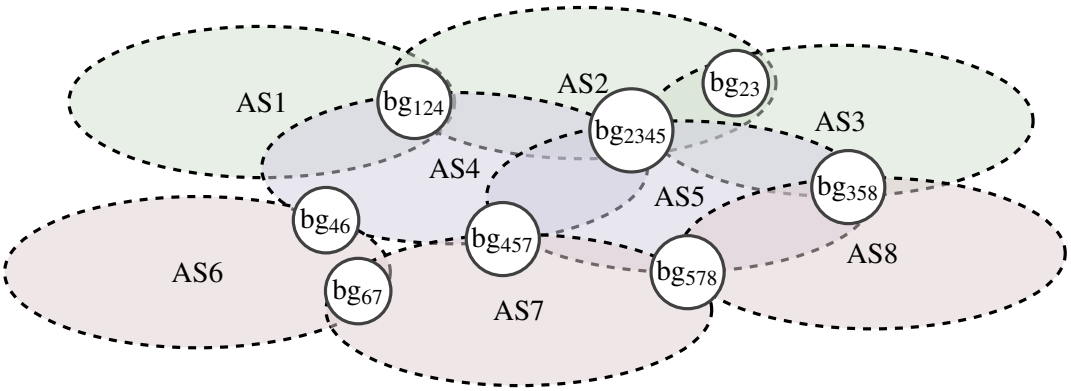


Figure 4.89: Autonomous network sections connected by gateway nodes

they need to be split and encoded into add-on envelopes. The real challenges reside in the control plane. Most transmission technologies bring their own control concept with them, typically an entire control protocol suite that is optimised to the key features and limits of the technology. To become exchangeable among ASs the control information used within a specialised protocol suit is reduced to a common minimal set of information, which is technology independently exchanged via standardised messages.

In this respect we should also mention the outsourcing of the path computation, utilizing specialised *path computation element*-(PCE) units located at central nodes reachable by all nodes of an AS with minimal latency. These open a utile, technology independent way for inter-AS communication, avoiding the need to translate control messages from one technology to the other at every gateway. Information exchange via PCEs, now also serving as BGP-gateways, also isolates the control planes form each other and implements information filtering as required for the privacy of the individual operators.

For example, a PCE can offer a limited number of path variants, together with predicted performance metrics for each, to a requesting neighbouring AS, which itself has a small set of path options and can now select the best fitting pairings and communicate these back to the AS it received the request from. Finally, the source AS picks the path that fulfils the quality requests stated by the virtual network management plane when signalling the routing request. Thereby *quality sensitive load balancing* can be realised, comparable to RSVP-TE [RFC3209], here entirely realised using the state awareness of PCEs, which considerably reduces the routing effort and complexity.

In addition, the virtual network also serves to reduce the stress on the physical resource management caused by fast changing traffic volumes and quality demands. The channels provided by the physical resources are traditionally optimised off-line to meet expected traffic volumes and quality demands. Because changing an optimised configuration always poses a high risk of service interruption and QoS failures, the optimisation is commonly performed far on the save side, intentionally overestimating the traffic volumes and service requirements, such that the solution can be in operation for a maximally long time. This is evidently not a very responsive approach and utilises the resource not maximally efficient. Anyhow, the *connections* configured according to this strategy define the so called *virtual network* that is actually used to transport data flows, independent if connection-less or connection-oriented. Thus, the links and connections of the virtual network are finally responsible for the actually achieved service quality.

Putting aside any possible technology constraints, the services that an ideal virtual network should today offer can be detailed for example as:

- **IP packet forwarding:** Evidently, the transport of IP packets is a must have criterion. How that is realised depends on the available features. In general, this basic IP forwarding service is to be performed in *best effort* manner.
- **Multi-service support:** The world is not digital, and so are the demands on data transmission quality not the same for all applications, even if these use digital data to perform their job. Thus, an ideal virtual network offers any transmission quality at any time. Of course this is not feasible. However, support and integration of *multi-protocol label switching* (MPLS) are rational demands nowadays. If in addition different technologies' ASs overlap each other, the hierarchic multi-layer architecture could be finally obsoleted.
- **Ethernet tunnelling:** Connecting remote LANs requires the encapsulation of their data-frames such that identical private IP-addresses do not cause problems when transported alongside traffic from other LANs. In particular, the *virtual private LAN service* (VPLS) [RFC4761, RFC4762] shall be supported. However, there exist many more protocols to perform similar tasks differently, and all these should be supported as well in order to not exclude customers that by chance own a device using a standard not considered.
- **Pseudo wire services:** Some applications need exclusive control over the timing of data transmission, for example to synchronise remote equipment. Commonly these need a time continuous connection that only poses constant propagation delays. To achieve mostly constant delays across a network the resources need to be strictly assigned, not only reserved. A limited jitter remains if physical resources are not assigned to exclusively serve the virtual line only.
- **Multimedia broadcast:** To efficiently transport identical data streams to many destinations no network resource should transport more than one copy of the same stream. To realise this the network nodes need to be able to effectively forward copies of a received data packet to any number of ports. Vice versa, for surveillance applications the reverse should be possible as well. If the same contents are to be transmitted with some time lag, distributed content caching at network nodes can save a lot of transmission capacity.
- **Self-scheduled connectivity:** Many potentially huge data transfer requests do not demand instant forwarding. For example, file transfers for backup purposes. These could be buffered and forwarded when the competition on the network resources is low. If enabled section wise, reusing the buffers provided for content caching, the network could most efficiently schedule the transmission of such delay uncritical elephant loads.
- **Impairment transparency:** To effectively support end-to-end QoS a potential bottleneck along a connection across the virtual network, which may occur dynamically, needs to be disclosed. Either to trigger re-routing to a feasible route bypassing the bottleneck, or to inform the application to cope with a temporarily reduced quality.
- **Autonomous open control:** When networks grow huge the control effort rises accordingly. A single control centre is not only a security risk, it also faces sever response time issues if multiple change requests arrive simultaneously. Therefore, a modern network control strategy should distribute the control burden as much as possible. At the same time, control shall be open to sensible overruling in order to serve special demands that exceed the capabilities of the implemented control mechanisms.

Many more services and features could be stated, and more will appear with ever increasing frequency as the applications plurality raises. A sustainable technology needs to be adoptable such that in can comfort future demands without a change in its core components.

Finally, to assure reliable network performance utile load control mechanisms need to be enforced. In particular, *overload prevention*. In that respect, the virtual network needs to have the capability to reject connection requests, or at least to drop flows at the ingress port if it cannot transport them, such that these flows never degrade other traffic flows.

4.4.3 IT-services stratum: application management

The top most management plane is also the most abstract and at the same time the one the user is most aware of. Its features directly influence the quality the user experiences. Therefore, this layer is responsible for the QoS delivery. Obviously, it cannot achieve this without the help of the lower layer management.

A problem is the distance form the network resources. Applications are managed by the users and content providers, not the network operators. Because latter are a heterogeneous group with individual interests and believes, commonly opposed to the demands and wishes of customers and providers, they have no natural right to control the applications and features thereof on offer. They have to serve them as good as they can and charge the costs caused.

To assure fair and persistent service provisioning and coverage, regulation bodies of several modern countries around the world specified some basic rules and targets alike those stated in [156]. These address in particularly network operators but need to be considered by equipment vendors and service providers likewise. Concerning network neutrality these were summarised in [130] to:

• no restrictions on the content accessed	*no content censure*
• no restrictions on the sites visited	*no origin censure*
• no restrictions on the platforms used	*no system binding*
• no restrictions on the equipment used	*no product binding*
• no restrictions on the communication mode used	*no protocol banning*

The rules seem natural, but business rarely cares for morality and therefore necessitates regulative intervention if a fair competition is missed. However, the quality that needs to be delivered is not regulated, only that it may not be artificially reduced for specific contents, sites, platforms, equipment, or protocols. Anyhow, the economic issues are not in the focus here, as is the persistent ignorance of major software vendors, which does cause platform dependent service quality issues.

The question is, how can applications be managed in support of the achieved QoS. At customer side this is primarily achieved by *hierarchic scheduling within the TCP stack*. Commonly the access link is the bottleneck, and thus it makes much sense to schedule uploads in a way that supports the required QoS per flow. At the customer side of a connection TCP and the layers above perform all the application management required.

On the other end of a network connection, at the servers offering the content to be downloaded, a similar approach can be implemented considering all customers currently running different applications. Here the actual traffic volume inserted can be modified to fit the capacity provided by the network, for example by applying different compression options. For example: video streams can be more or less compressed, web-page contents are sent in order of importance, background music last, it is likely suppressed anyhow, and the encoding of the data stream can include redundancy in order to mitigate lossy connections. All this does have an effect on the QoS, but to obey the rules for fairness, it needs to be applied equally for all customers and services.

To utilise the available options for application management it is necessary that the interface between the applications and the virtual network supports the exchange of the thereto required information:

- **application –> virtual network:** Advertise the requested service
  - LOAD CHARACTERISTICS: Expected mean and variance of the load to be transported, for both, inter-arrival times and sizes of load units (e.g.: min/max, on/off, Poisson, Pareto)
  - REQUESTED QOS: Minimal throughput, maximal delay and jitter bounds to be met for intended service delivery quality (QoE)

  - FAILURE STRATEGY: What to do if requested QoS cannot be achieved (e.g.: increase throughput and/or QoS, deliver as it is, quit delivery)
- **virtual network –> application:** Report upon the delivered service
  - SERVICE CHARACTERISTICS: achieved packet rate/loss/delay/jitter (statistics enabling the application to sensibly adjust the load presentation toward better QoE)
  - PERFORMED ADJUSTMENTS: session layer configuration changes (feedback allowing the application to learn what in the moment is required to deliver the current QoS)

The information from the application to the network upon the requested service, and vice versa from the network to the application concerning the fulfilment of the performance targets stated by applications, needs to be tunnelled through layers 4 to 6 in order to reach the appropriate management domain. This does not mean that these layers become obsolete. In contrary, they actually provide the functionalities to change an applications ingress load in a way that fits the actually available QoS or can be used to improve the delivered QoS. However, due to the plurality of applications, these layers cannot decide on their own how to best change the presentation and scheduling of an applications data load. Only the application and virtual network management can decide this. Thus, the application needs to tell the layers 6 how to present the data flow, and thereto it needs the feedback from the virtual network. The layer 5 can be used by the network to adjust the load insertion such that it can deliver the QoS requested by the application. The best session layer adjustment depends on the network's current state and therefore the adjustment needs to be decided by the virtual network.

Thus, in addition to the interface between the application and the virtual network we also need the following one-way interfaces to the presentation and session layer:

- **application –> presentation layer:** Configure/adjust the content encoding options
  - ENCODING REDUNDANCY: increase/decrease the amount of redundancy
  - COMPRESSION FACTOR: increase/decrease the content compression
- **virtual network –> session layer:** Adjust load insertion and reassembly options
  - FLOW PRIORITISATION: Increase/decrease/switch the flow's priority/traffic class
  - SCHEDULING & BUFFER SIZE: How to adjust the load insertion and reassembly (e.g.: session prioritisation, load fragmentation – packet size, reassembly buffer size)

Concerning the session layer adjustments it needs to be noted that the virtual network should not be empowered to perform any adjustments. The range opened for network layer initiated adjustments shall be pre-configured per application, primarily to mitigate the risk of runaway configuration in case two applications compete for resources that are insufficient to serve both. On the receiver side, the session layer is the instance that can measure the end-to-end QoS, which shall be reported to the virtual network management in order to initiate adjustments where required. The same applies in principle for the application layer at receiver side in respect to calculated QoE metrics.

The information flow among layers and management blocks is sketched in figure 4.90. The terminals and layers above reside within customer equipment, and thus also the application management. If coordination among the local application management blocks schall be provided across the network, some networking instance needs to be provided, which evidently shall be realised distributed, for example as overlay network. Noteably, the virtual network management is a network service that cannot be located to customer premises. Still, it needs to communicate with the local session layers to learn about the delivered QoS in order to sensibly adjust the prioritisation for best possible QoS delivery.

In theory both, the application management as well as the virtual network management, could initiate a re-routing in case the delivered QoS becomes insufficient. However, the virtual network management does not know if the application actually demands this. Thus, re-routing shall only be initiated by the application management. Actually, a make before break policy realising a mostly

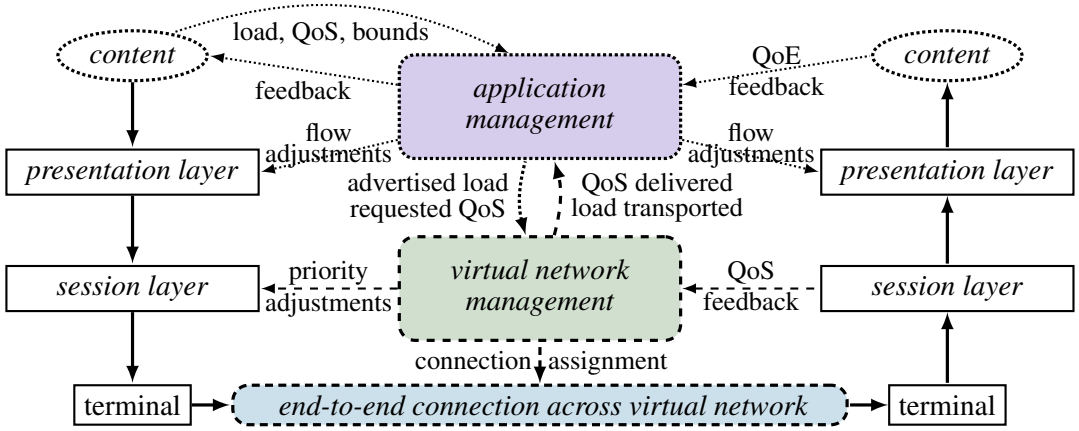


Figure 4.90: Network feedback to applications for content sensitive response

seamless *hand over* is possible. The decision sovereignty of the application layer upon the path quality required and the coutermeasures to be taken in case of underfulfilment becomes self evident if the customer can set a least performance limit. In case this cannot be reached, the customer prefers not to use the application. For example, a video or audio stream that cannot be received with sufficient continuity is likely not required at all. Therefore, any attempt to transport the stream is a wast of resources only.

Finally, we note that the application control can reside in the terminal equipment, for example a set-top-box, whereas the presentation layer functionality, being decoding & decompression, may be outsourced to a video screen. Even the entire layer 4 to 6 stack may be outsourced if the monitor is connected via a LAN. Such situations become increasingly popular with the rise of so called *'networked consumer equipment'*.

4.4.4 Lean connectivity provisioning

In economics it is perfectly accepted that bureaucracy is an efficiency trap. To maximise efficiency first the communication among systems needs to be minimised to the essentially needed. Never should any entity be misused to only forward information, neither to process an issue twice. Optimally, communication shall occur only among the entities that either generate or use the exchanged information. More technically, we call the information exchange an *interface*, which typically is specified independently for the two possible information exchange directions.

To realise lean management it is necessary that the demand interfaces are directed from the customer toward the resource, such that step by step the request fulfilment happens in the opposite direction without causing any delays. For a production line this means that the order of a customer is served by the delivery department, which takes the finished product from the storage at the end of the production line. The now empty slot in the storage triggers the end-production to finish a unit of the now missing kind. To do so some parts from the storage in between them and the next upstream production step on the line are used, leaving gaps there, which informs the next production unit and so on, till the order of raw materials is reached.

The key elements controlling the entire process are the storages in between the production steps. These need to be optimally designed, which is only possible based on demand predictions. Best possible predictions are a joint task of the entire management team. Therefore, in terms of network management, all units that perform decisions need to monitor the load and predict the demand. To not fail in case the prediction happens to be below the actual demand, the units always need to reserve some capacity. To be economic this reserve shall be kept minimal.

Adjusted to our three layer network management hierarchy, this results in the process sketched in figure 4.91. The service stratum takes an available connection from the virtual network and activates

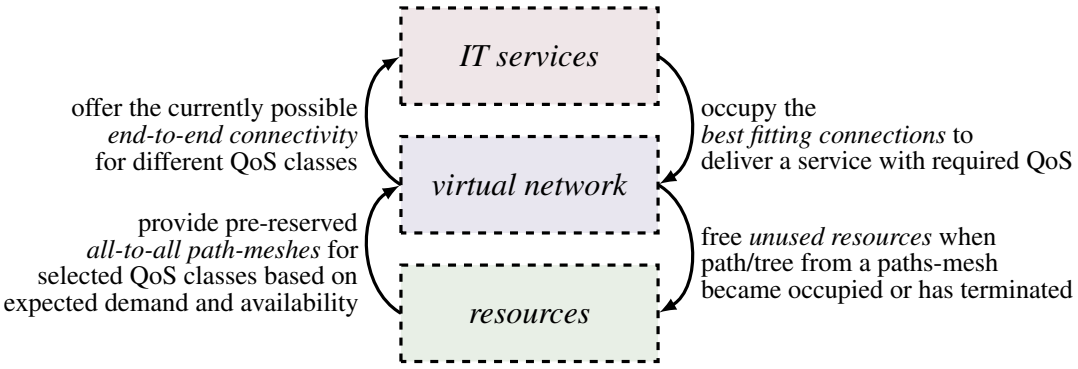


Figure 4.91: Network management based on QoS sensibly provisioned and advertised connectivity

it. In response the virtual network prepares a new connection of the kind just taken away to have one in spare again. The preparation of the new connection assigns some links provided by the physical network, which in response prepares new links by assigning the required resources.

If implemented well, this provisioning process is entirely self controlled and the entities are entirely autonomous in respect to how they provide what the downstream entity demands. Vice versa is the demand communicated from the application to the resources without disclosing more than the necessary information.

The key elements to achieve this lean network management are maximally autonomous, smartly distributed, management domains. The virtual network, being the glue in between the resources and the applications, has in this respect the most important role: it fulfils the customer demands or rejects them if the demands cannot be fulfilled. As already stated, to perform this task effectively, the virtual network needs to have different connectivity options at hand.

An option to provide this plurality is MPLS, realised on both levels, inter-AS and intra-AS. However, the configured paths within an AS need to report their quality in order to allow the calculation of end-to-end QoS for the inter-AS LSPs. These are in principle required from any node to any other, and especially in between all gateways. In addition, the paths need to provide a plurality of qualities that fits to the expected traffic demand risen by the applications. To calculate this explicitly may become exhaustive.

A simpler solution thereto are dummy loads used to assess the QoS per intra-AS connection configured and not yet occupied. Note, we assume here an all-to-all routing algorithm that proposes a paths-set that in itself may re-use resources. These paths are only an offer and once a path is activated, the not used resource become freed and a new set of all-to-all paths is calculated. Evidently, the amount of pre-configured intra-AS connections shall be minimised to keep the amount of reserved but currently unused resources reasonable. However, an AS less loaded than neighbouring ASs shall always offer more connectivity than an AS loaded more when its neighbours are (*swarm control*).

On the inter-AS level, travelling agents (ants) shall collect and recommend different combinations of intra-AS connections in order to provide a maximally diversified variety of connections to the virtual network management. Their number shall reflect the expected request arrival rate and control domain diameter, such that in average the information is regained before another request for the same terminal nodes occurs.

5 Conclusions

In the course of preparing the grounds for this PhD-thesis we submitted a project proposal on modelling the control plane for actively switched transparent optical access networks. The proposal was rejected three times till it was too late to run the project in due time. Assumedly the reasons for rejection refer to what one and the same reviewer stated in his review: *"The duration of the project* (30 months) *could be too long with respect to the parallel technology evolution"*, referring to the Carrier Ethernet evolution, and *"The use of the analytic approach is a real challenge, it is hard to foresee the real effectiveness of this approach"*, referring to the *"unavoidable simplification assumptions"* and *"computational complexity too high to study real network scenarios"*.

On an alternate project proposal targeting *traffic conditioning* in favour of improved transport quality, twice rejected before giving up, we got the following reply: *"The obvious problem is the size of the chain and the fact that it is often not possible to compute by standard methods solutions for systems as complex as the ones proposed. . . . Whether this is possible is an open question and . . . "*

These replies raised some doubts on the grounds of the thesis: If a simple all optical access network is by an expert assumed intractable for mathematical models, how should it be possible to model paths across several network ranges? And may it be truly impossible to solve lengthy chains of flow aware queueing systems? In the end, we admit: *a too detailed model is rather useless in respect to understanding and reliably tackling the true issues.* Too many effects interfere and these are better handled one-at-a-time, node-by-node, instead of designing an exaggerated model encompassing all possibilities in a single state transition diagram.

When searching for new textbooks on queueing models at a global web-store we happened to stumble over the following statements (excerpts) from a user commenting on the offered textbook: *"The statement* 'The world does not look like an *M/M/1* queue!' *possibly yields the best of all possible answers to* 'Why the dearth of books that approach computer performance from the standpoint of queueing theory'. *. . . One must resort to numerical methods . . . use stochastic discrete event modelling and simulation."* Again, not a very promising finding while writing a PhD thesis on the use of queueing models to predict the performance of multi-service data networks.

Finally, we also found in a white paper presenting a multi-service network node of a major vendor, not to be disclosed here, a recommendation proposing to *optimise by trial and error, one feature/node at a time only*. Reading this in a paper affiliated to a major vendor expresses the complexity of the problem and its potential to exceed tractability. In the configuration manual for a similar switch we found: *"Only certain classes of service of traffic can be flow controlled while other classes are allowed to operate normally."* This contradicts the common approach with queueing systems, where specific flows become privileged. The potential discrepancy in equipment and queueing system designs questions the utility of results achieved using queueing theoretic analysis.

Besides, the vast plurality of individual mechanisms and configuration options complicate a well planned, cognitive, by queueing model approved parametrisation. This may explain the operator attitude, widely advisable for systems exceeding human cognition: *never change a running system*. Better set-up and excessively stress-test a new one before touching a satisfactorily operating one.

End-to-end analysis

The initially intended solution of the entire end-to-end path sketched in figure 5.1 is found to be in theory possible, at least for finite queues, but useless because of the many unknown parameters to be individually evaluated or guessed. It is not feasible to analyse every variation of possible background

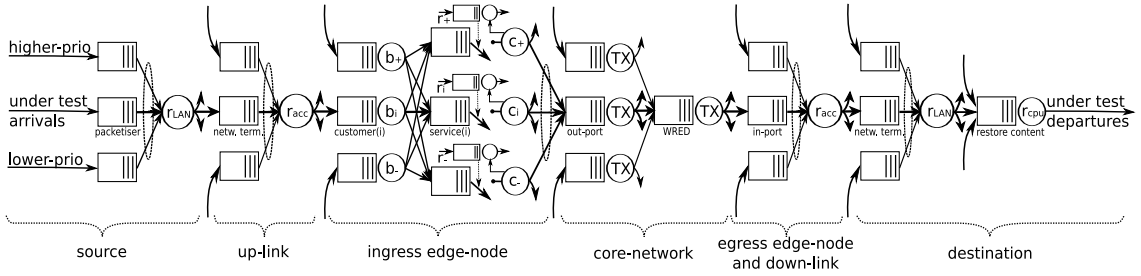


Figure 5.1: Chain of queueing systems contributing to a connection under test

traffic combinations and characteristics (distributions) in order to identify the worst case for the connection evaluated. Also, for every connection not sharing the same resources an individual model needs to be designed and evaluated, such that the effort to analyse an entire network section appears rather intractable. Besides, if the queues are huge and the number of flows is high, we commonly run into numeric calculation problems and excessive memory demand. Using the *sparse matrix* option, provided for example by *Octave* [29], we experienced that the provided numeric algorithms quite regularly give up prior reaching a solution when the state space is huge, even for rather simple, hardly populated state transition matrices Q.

In the introduction we introduced the *network calculus* (section 1.3). Besides being incompatible to the unbounded distributions commonly used with queueing models, the achieved bounds delivered by this method are in general too far apart to enable reliable performance prediction. Truly, the lower bounds can be used to specify service level minima and the upper bounds to highlight the potential of a network architecture. However, the longer the paths are, the more apart become the bounds. In general the *mean performance* is more important than the absolute bounds. This, the deterministic network calculus cannot offer, and the *stochastic network calculus* has been found too complex to be useful for non mathematicians.

Lessons learned

In retrospective we come to the conclusion that if we look at communication networks in a purely bottom-up approach, analysing protocols, procedures, and mechanisms one after the other, we likely become lost in the ingenious details and the unbound variety of the many involved mechanisms. If we look at communication networks in a pure top-down fashion we always find the OSI layers, and the implicit *lower layer serves the layer above* scheme. We clearly identify the layer internal mechanisms that realise the layer by layer provisioning and understand their demand. However, the OSI model does not yield answers to the question why a network behaves as it does in certain circumstances. To understand these we need to understand the procedures realised by the involved mechanisms and, most importantly, their side-effects. Consequently, we should be aware that whenever we change one detail in order to change a certain behaviour, this will have effects on other behaviours, possible in other circumstances we currently might not think of.

This complexity of interrelations recommends the use of control strategies that are known to be insensitive to imprecise knowledge and particularly applicable for complex dynamic systems. True, perfect knowledge of all network states is possible, theoretically. However, the traffic dynamics and the plethora of mechanisms that respond to changes causes that absolute network states need to be assumed outdated once gathered. Statistical measures are more persistent, but never reflect the

current state precisely. Thus, using statistical measures for control decisions precludes perfect results. Imprecision within the data-base renders precise system models exaggerated. The comparison of queueing models shows that rather simple models yield results quite close to those of precise models, closer than the deviation that results from imprecise state assumptions. Thus, simple models very likely offer equally good but far less complex system approximations in practice.

Final approach

In the end, we discussed system models of the mechanisms used at different locations within a network one-by-one. Intermediate nodes provide differentiated scheduling (section 4.1) and congestion avoidance mechanisms (section 4.2), which possibly may be configured to a flow's needs. Edge nodes commonly implement some kind of ingress control that limits the instantaneous ingress traffic (section 4.3). A shared medium in the access network commonly adds scattered resource provisioning, controlled by some polling strategy. Terminals implement transport control mechanisms that adjust the flow characteristics (arrival process) to maximise the throughput, or in other words, that try to minimise the time needed to transport data from the source to the destination.

In case of loss-controlled transmission, load units may become re-transmitted in case they had not been successfully transmitted in the first attempt. This principally adds delays. Firstly, we have to consider the time-lag in between transmitting the last bit of a load and receiving information on the transmission success. Secondly, the transmission channel may be occupied at the time we receive the information that a load needs to be re-transmitted. The remaining service time needs to be added in order to consider the time we have to wait until the channel becomes available for the re-transmission. The *correlation* of arrivals thereby introduced is commonly not considered because we commonly assume negligible loss-rates.

Identified challenges

Primarily the multitude and variability of the mechanisms that are actually involved along an end-to-end data connection, causes an exact model to represent a special case only. In addition, such detailed models would be very complex and in general not analytically solvable. Only for homogeneous cases we know theories and approaches that yield close-form solutions. To derive general results from special cases is in principle not possible. Therefore, we did not attempt to define analytic models that join flow control, medium access, per hop behaviour, and feedback signalling mechanisms. Instead, we modelled and analysed the different mechanisms individually and based thereon, made an educated guess upon any interrelations, in order to derive recommendations for a potential future network management architecture.

Besides, we glanced on autonomous network control. The primary problem of network control is the absence of precise knowledge. This results from the stochastic traffic insertion, the distribution of control mechanisms and transport resources, and the information propagation delays thereby caused. We intuitively favour *swarm intelligence* to best mitigate this problem. However, a proof is out of the scope here. Still, it is well known that precise knowledge is not necessarily required for local decisions to suite global targets.

The prime challenge is to identify the information bits locally required and to find a rules-set that dynamically serves the global targets. The methods to realise a *smart Internet* are readily available, but a convincing concept is missing. Possibly because engineers tend to look for an everlasting solution, whereas nature tells very clearly that only self-learning adaptive systems are truly sustainable. In that respect we surmise that approaches alike the *Residual Network and Link Capacity* algorithm [157] can be applied section wise, management free and with reasonable computation effort.

A Addenda

The addenda contain content, derivations, and proofs that are either too general or too specific to be included in the main text, commonly. Here, the common derivations were briefly included in the course of the thesis because they are intended to present approach variants that the reader may apply with similar problems. More detailed proofs and discussions on the mathematical background of the presented approaches can be found in the many textbooks well known, in particular [14,34,57,62,85]. Thus, chapters 2 and 3 could have been entirely placed in the addenda because they present well established knowledge only.

Besides a nomenclature summary and the abbreviations list the addenda contains only a brief introduction and presentation of some core program code examples used in preparing the thesis. These here included routines comprise all the elements required to detail the approach, but they lack supplementary routines not required to grasp the idea.

A.I Program code

Please note that in this section exemplary snapshots of *living* working codes are presented, meaning non-professional pieces of program code that are not indented to be used without proper understanding. The presented codes are neither fail-proof, nor can it be guaranteed that they still deliver the results presented in the course of this theses. They should, but having been extended and refined case by case, they were not checked against examples done prior the change, hence called a living code. However, in case of major changes the routines were re-named, such that the routines used for earlier examples remain available unchanged.

Most examples shown throughout the thesis rely on three specific parts. An m-script that sets all parameters, controls the evaluation over different parameter ranges, and finally plots and saves all gained results. This script commonly calls the other two key routines, the simulation core and the calculation function, if latter is not coded directly within the script. Several more subroutines (m-functions) are used in particular by the simulation core, for example to generate one-by-one the required random numbers according to different distributions. This allows on-the-fly generation, eliminating the need for recorded random number sample traces.

The entire collection of routines developed in the course of this thesis constitute a so called *university tool*, which may be shared for non-commercial use only, and solely on own risk, without any liability of the implicit copyright holders. Without written consent personally signed by the author any commercial use of either, the entire collection of routines, subsets or individual routines, or any code segment, is explicitly prohibited. Primarily because the author himself cannot guarantee the code to be free of code segments owned by third parties. Only the routines outlined in the regular text are made available for public use, constraint to being correctly cited and openly provided.

A.I.1 Example script (m-code) to generate plots

An exemplary script to generate the plots presented in the course of the thesis is presented rather comments-less. The procedures included are rather simple, such that a detailed explanation is surplus for a reader with some programming skills. A detailed presentation for a novice programmer is

out of the scope here. Please refer to the *octave* help (`help <command>`) and documentation [29] for common usage and details on the used *octave* commands. More information provided by the open *octave* programmers community can be found on-line, for example searching for `Octave <command>` and skipping the many copies of the help content, not providing any more information than the built in *octave* help.

Some routines used in this script not being *octave* commands, for example `setMyColors`, `errorPlot`, provide slight refinements of existing *octave* commands mostly for consistency and a personalised appearance. They are not related to the topic of the thesis and thus not explained. The code presented is a more or less a 1:1 copy of an actually used script, including rudimentary comments intended as reminders to the author himself. Still, surplus commentary lines have been deleted.

Perform a simulation study and generate plots of results

```
%DPS example - weights in second c-column
%  function [Q, res, statis] = GGpscgSim(a, Arr, b, Serv, n, s, c, pol, loop, trans, histp)
%  statis(1:fnum,2:10)=[mA(:),ebA(:),cvA(:),mD(:),ebD(:),cvD(:),mB(:),ebB(:),cvB(:)];
%  statis(1:fnum,11:19)=[mX(:),ebX(:),mQ(:),ebQ(:),mF(:),ebF(:),mW(:),ebW(:),cvW(:)];
pageStdOut=page_screen_output; if pageStdOut more('off'); endif
nh=!ishold;

loop=200000; trans=50000; ints=40; histp=0;
dists={['nExp';'Det'],['nExp';'Hyper']}; %dists={['nExp';'nExp'],['nExp';'Det'],['nExp';'Hyper']};
distn=length(dists); pol=['RR';'FIFO'];
mr=1; rho=linspace(mr/ints,mr-mr/ints,ints-1); myticks=[0:0.1*mr:0.9*mr];
mytitle=['MGpspSamplesA',datestr(date,'yymmdd')];
ri=[0.25,0.25,0.25,0.25], fn=length(ri);
gi=[1,2,4,8], %gi=[1,2,4,8]; gi=[1, 1+1/2, 1+1/2+1/3, 1+1/2+1/3+1/4]; gi=[1,3/2,11/6,25/12];
ci=Inf(fn,1); %ci(:,1) has to be 'Inf' for Erlang setting
n='ps'; s=inf; c=[ci(:),gi(:)]; %weights are added to customers parameter

if length(ri)!=length(gi) error('loadsplit and weights vectors do not match in size'); endif
samples1=NaN(length(rho),1+2*distn*(fn+1)); % (fn+1) -> results for sums
samples1(:,1)=rho;
samples2=NaN(size(samples1));
samples2(:,1)=rho;
samples3=NaN(size(samples1));
samples3(:,1)=rho;
for j=1:distn
   arr=dists(j){}(1,:);
   ser=dists(j){}(2,:);
   jof=1+2*(j-1)*(fn+1);
   for i=1:length(rho)
      a=ri(:).*rho(i); a=[a,5*ones(size(a))]; %display(a); %a=ri(:);
      b=1; b=[b,5*ones(size(b))]; %display(b); %b=1./rho(i);
      [Q, res, statis] = GGpscgSim2(a, arr, b, ser, n, s, c, pol, loop, trans, histp);
      samples1(i,jof+1:2:jof+2*(fn+1))=statis(:,11)';
      samples1(i,jof+2:2:jof+2*(fn+1))=statis(:,12)'; %system filling
      samples2(i,jof+1:2:jof+2*(fn+1))=statis(:,15)';
      samples2(i,jof+2:2:jof+2*(fn+1))=statis(:,16)'; %mean flow time
      samples3(i,jof+1:2:jof+2*(fn+1))=statis(:,4)'; %arrival cov
      %samples3(i,1+jof)=statis(1,19); %waiting cov
      samples3(i,jof+2:2:jof+2*(fn+1))=statis(:,7); %departure cov
   endfor
endfor
if 1 samples={samples1,samples2,samples3}; save([mytitle,'.dat'],"samples"); endif
if 1 %plot simulation results
   figure(1); errorPlot(samples1,0); axis([0,1,0.005,11]); title([mytitle,' E[X]']);
   figure(2); errorPlot(samples2,0); axis([0,1,0.5,200]); title([mytitle,' E[T_f]']);
   figure(3); setMyColors; plot(rho,samples3(:,3:2:end),'.x');
   axis([0,1,0.05,6.75]); title([mytitle,' \cov \vartheta']);
endif
if 1 %calculated E[X] and E[T_f] values
  mr=min(1,mr);
  steps=4*ints; rhos=linspace(mr/steps,mr-mr/steps,steps-1);
  Tf=MMpspCalc(rhos,ri,gi); %call analytic equation
  Xmm1=rhos./(1-rhos); %calculate it straight away
  figure(1); hold('on');
```

```
    plot(rhos,Xmm1(:),'color',getColor('336699'),'linewidth',1.5);
    if nh hold("off"); endif
  figure(2); hold('on');
    semilogy(rhos,Tf(:,1:end-1),'color',getColor('009900'),'linewidth',2);
    semilogy(rhos,Tf(:,end),'color',getColor('00cc00'),'linewidth',2);
    if nh hold("off"); endif
endif
if 1 %save final figures to files
  print(figure(1),[mytitle,'_X'],"-color","-dsvg");
  print(figure(2),[mytitle,'_Tf'],"-color","-dsvg");
  print(figure(3),[mytitle,'_cD'],"-color","-dsvg");
endif
fflush(stdout);  if pageStdOut more('on'); endif
```

Note, this is a working script intended to be adjusted run-by-run in order to variate the parameters such that different studies are performed. The included [`if 1 ... endif`] clauses enable a simple exclusion of segments during refinements. Code embraced by [`if 0 ... endif`] is never executed.

Being a script, the code does not return any variables, it calculates everything in the current scope, such that after the script has been executed, all variables created and manipulated are still available. Nice for debugging, this may cause problems if for example a variable definition is commented out in the code but the variable remains defined in the current scope. Use the `clear all` command in between running the script to be sure no obsolete variable and function assignments remain valid.

A.I.2 *G/G/n/s/c/policy* simulation procedure (*m*-code)

The simulation core routine (coded as a *function*) is presented without extensive textual explanations. Please refer to section 3.4 for step-by-step details on the different parts and functionalities. Rudimentary comments and thoughts of the author are included in the code, which represents a slightly cleaned 1:1 copy of the code actually used toward the end of the studies presented in the course of the thesis. Only surplus commentary lines and no more used code segments (*dead code*) have been deleted.

G/G/n/s/c/policy

```
%simulate a G^G/G/n/st/cg/xxx-system - i.e. one load point
%note: 'a' specifies load per customer if 'c'<Inf, else the total load per flow
%scalar [s,t]=RED, [s,t]+g-vec=WRED, s=t-array=thresholds, vector [s,t]=perFlow RED
%this version uses dynamic structure to record states

function [statis] = GGGnsdtbrdcgSim(a, Arr, b, Serv, n, s, c, pol, loop, trans, histp)

%set defaults if not all parameters are given
  if (nargin<1) a=0.5*[0.2;0.3;0.5]; endif  %three flows M/M/3 at load=0.5
  if (nargin<2) Arr=['nExp';'nExp']; endif  %M^M/M/3 at load a
  if (nargin<3) b=1/3; endif                %G/M/3 at load a
  if (nargin<4) Serv='nExp'; endif          %G/M/3 at load a./b
  if (nargin<5) n=3; endif                  %G/G/3 at load a./b
  if (nargin<6) s=9; endif                  %G/G/n at load a./b
  if (nargin<7) c=Inf; endif                %G/G/n/s at load a./b
  if (nargin<8) pol=['RR';'FIFO']; endif    %G/G/n/s/c at load a./b
% if (nargin<8) pol=['ES';'FIFO']; endif    %G/G/n/s/c/ES at load a./b
  if (nargin<9) loop=4000; endif            %G/G/n/s/c/pol at load a./b - too short loop for good results
  if (nargin<10) trans=ceil(loop/2); endif  %G/G/n/s/c/pol at load a./b - sufficient for feasible loop
  if (nargin<11 || ~histp || any(histp==0)) %do not plot histograms x=0-10 with 75 bins if not provided
    histp=[10,75]; draw=false; else draw=true; endif
  if (isscalar(histp) && histp==1) histp=[10,75]; %draw with default histplot parameters
  elseif isscalar(histp) histp=[histp,75]; endif  %draw with default bin number

%parameter adjustments
  if ~isscalar(n) && (min(n(1:2)=='ps') || min(n(1:2)=='PS'))
    ps=true; n=1; %more processors, e.g. 'ps3', ?useful? not implemented!
  elseif ~isscalar(n) error('restricted server occupation per flow not implemented yet!');
  else ps=false; endif
  if (rows(Arr)==1 && rows(a)>1) sArr=Arr; %same Arrival process for all flows
```

```
    for i=2:rows(a) Arr=[Arr;sArr]; endfor
  elseif (rows(Arr)==2 && rows(a)!=2) sBat=Arr(2,:); sArr=Arr(1,:); Arr=[]; Bat=[];
    for i=1:rows(a) %same Arrival and Batch process for all flows
      Arr=[Arr;sArr]; Bat=[Bat;sBat]; endfor
  elseif (rows(Arr)==2*rows(a))
    Bat=Arr(2:2:end,:); Arr=Arr(1:2:end,:); %split Batch and Arrival distributions
  elseif (rows(Arr)!=rows(a)) error('Arrival/Batch process definition problem.'); endif
  if (rows(b)==1 && rows(a)>1) sb=b;
    for i=2:rows(a) %extend unique service vector to flows number (set same for all flows)
      b=[b;sb]; endfor endif
  if (rows(Serv)==1 && rows(b)>1) sServ=Serv;
    for i=2:rows(b) %extend unique Service process to flows number (set same for all flows)
    Serv=[Serv;sServ]; endfor endif
  if (rows(c)==1 && rows(a)>1) sc=c;
    for i=2:rows(a) %extend scalar population to vector (same population per flow)
      c=[c;sc]; endfor endif
  if (columns(s)==3 && rows(a)!=3) tbs=s(:,3); tar=s(:,2); s=s(:,1); %token bucket parameters extracted
  elseif (columns(s)==3 && rows(a)==3) ...
    warning('ambivalent parameters -> non token-bucket case is chosen!'); endif
  if (columns(s)==rows(a) && rows(s)==1) comqu=true; s=s'; %array=>shared queue with thresholds
  elseif (columns(s)==2*rows(a) && rows(s)==1) comqu=true; %random drop
    thlow=s(1:2:end)'; s=s(2:2:end)'; thlow(find(thlow==s))--; %thlow=letin, s=block
  elseif columns(s)>1 comqu=true; redths=s(:,2:end); s=s(:,1); %split size and threshold infos,
  elseif rows(s)>1 comqu=flase; %usual multi queue system (s-vector)
  else comqu=true; endif %scalar s (multiple flows rows(a)>1)=>usual shared queue
  %if (columns(c)>1 && rows(c)==1)
  c=c'; endif %convert single row into a vector, scalar=>joint population not realised!
  if (columns(c)>1) g=c(:,2:end);
    c=c(:,1); else g=[]; endif %extract weights etc. from number of customers vector
  prevSigm=[];

%parameter checks
  if (rows(a)!=rows(Arr))
    error('\n SORRY: different size for arrival (%d) and Arrival process (%d) vector is not possible.',...
      rows(a), rows(Arr)); return;
  elseif rows(a)!=rows(b)
    error('\n SORRY: different size for arrival (%d) and service (%d) vector is not possible.',...
      rows(a), rows(b)); return;
  elseif rows(b)!=rows(Serv)
    error('\n SORRY: different size for service (%d) and Service process (%d) vector is not possible.',...
      rows(b), rows(Serv)); return;
  elseif(rows(s)>1 && rows(s)!=rows(a))
    error('\n SORRY: different size for arrival (%d) and queues (%d) vector is not possible.',...
      rows(a), rows(s)); return;
  elseif(rows(c)>1 && rows(c)!=rows(a))
    error('\n SORRY: different size for arrival (%d) and population (%d) vector is not possible.',...
      rows(a), rows(c)); return;
  elseif(min(a(:,1))<eps) error('\n SORRY: a simulation with arrivals=%3.2f is not possible.',a); return;
  elseif (min(b(:,1))<eps) error('\n SORRY: a simulation with service=%3.2f is not possible.',b); return;
  endif
  fnum=rows(a);         %number of flows
  if (loop<trans)
      fprintf('\n WARNING: transient phase (%d) exceeds loop (%d) -> loop changed to %d.\n',...
        trans,loop,loop+trans); loop+=trans; endif
  if(n<1) fprintf('\n WARNING: %d servers cannot be simulated ->  set to n=1.',n); n=1; return; endif
  if rows(pol)==1 %policy shortcuts
    if isscalar(pol) switch pol
      case 1; pol=['RR';'FIFO']; case 2; pol=['RR';'LIFO']; case 3; pol=['RR';'RAND']; %round robin
      case 11; pol=['SP';'FIFO']; case 12; pol=['SP';'LIFO']; case 13; pol=['SP';'RAND']; %strict priority
      case 21; pol=['maxQ';'FIFO']; case 22; pol=['maxQ';'LIFO']; case 23; pol=['maxQ';'RAND'];
      case 31; pol=['minQ';'FIFO']; case 32; pol=['minQ';'LIFO']; case 33; pol=['minQ';'RAND'];
      case 41; pol=['ES';'FIFO']; case 42; pol=['ES';'LIFO']; case 43; pol=['ES';'RAND']; %egalitarian sharing
      case 51; pol=['PS';'FIFO']; case 52; pol=['PS';'LIFO']; case 53; pol=['PS';'RAND']; %egalitarian sharing
      case 61; pol=['WRQ+';'FIFO']; case 62; pol=['WRQ+';'LIFO']; case 63; pol=['WRQ+';'RAND']; %WFQ approx.
      otherwise error('a policy shortcut %d is not implemented,...
        please specify policy explicitly ["RR";"FIFO"]',pol); return
      endswitch
    elseif sum(all(pol==['FIFO';'LIFO';'RAND'],2)) pol=['RR';pol];
      %the all(..) may cause the "warning: mx_el_eq: automatic broadcasting operation applied"
    else pol=[pol;'FIFO']; endif; %add missing default
  endif
  switch deblank(pol(1,:))
```

```
    case 'ES'; es=true; pol=['RR';pol(2,:)]; ps=false; %egalitarian sharing
    %case 'DES'; es=true; pol(1,1:4)='RR  '; if ~any(g) g=ones(fnum,1); endif, ps=false; ...
        % discrimatory sharing - not implemented?
    case 'PS'; ps=true; pol=['RR';pol(2,:)]; if ~any(g) g=ones(fnum,1); endif, es=false; ...
        % processor sharing (DPS if g_i !equal)
    case 'DPS'; ps=true; pol=['RR';pol(2,:)]; if ~any(g) error('DPS needs weights'); endif, es=false; ...
        % discrimatory processor sharing
    case 'WFQ'; pol=['WRQ+';pol(2,:)]; warning('WFQ is not precisely implemented -> approximated by WRQ+'); ...
        ps=false; es=false;
    otherwise ps=false; es=false; %neither
  endswitch
  % for ES the serving order equals RR; but servers may be assigned in parallel, when available!
  %if isscalar(s) display([a,b,c,g]); else display([a,b,s(:),c,g]); endif %display parameters

%const definitions and initialisation
  mout=sum(a(:,1).*n.*b(:,1))/sum(a(:,1));  %calculated mean service rate (fair serving):
  rhoi = a(:,1)./(n*b(:,1)); l=sum(rhoi);   % mean load = sum(arrival/service-rate) over all flows
  for i=1:fnum      % correct for Engset case
    if (c(i)<Inf) rhoi(i) = c(i)*a(i,1)/(a(i,1)+b(i,1)); l += rhoi(i) - a(i,1)/(n*b(i,1)); endif
  endfor
  fprintf(' Load=%5.4f | Arr1: %s / Serv1: %s | Servers=%d | Size1=%d | Samples=%d \n',...
    l,Arr(1,:),Serv(1,:),n,s(1),loop);
  loop+=trans; %add transient phase to loop
  if isscalar(c) cm=c*ones(fnum,1);
  else cm=c; endif %maxpopulation vector (to be deduced by sum(X), not X(fi)!)
  darr=[]; %used in Engset case for delayed arrival generation due to no customers left
  if length(histp)<2
    fprintf('\n Warning: histogram parameters were not provided as expected \n...
        -> set to default tmax=10, bins=75');
    histp=[10,75]; endif
  if any(c<inf) dispc=c; else dispc=[]; endif, %keyboard(); %dispc=zeros(length(g),0);
  if ~exist('redths') || isempty(redths) redths=[]; endif
  if exist('thlow') rdrop=zeros(fnum,max(s)+1);
    for i=1:fnum rdrop(i,:)=([0:max(s)]-thlow(i))/(s(i)-thlow(i)); endfor %set drop rates per state per flow
    rdrop(find(rdrop>1))=1; rdrop(find(rdrop<0))=0; %display([thlow,s,rdrop]);
  endif
  if exist('thlow') display([thlow,s,dispc,g]);
  elseif rows(s)>1 display([s,redths,dispc,g]);
  elseif rows(g)>1 display([redths,dispc,g]); endif
  fflush(stdout); pageStdOut=page_screen_output;
  if pageStdOut more('off'); endif  %set stdout to NOT wait when screen is full

%Local variables definitions and initialisation
  Waiting=cell(fnum,1); %waiting time arrays for queued customers (required for FIFO,LIFO,...)
  %inter-event times (in = till next arrivals, out = till next departure, t = since last event)
  if (isscalar(c) && c==Inf) infci=1;
      in=zeros(fnum,1); %initialise first arrivals at 0
  else infci=max(c(find(c<Inf))); if any(infci) infci.+=1; else infci=1; endif
      in=Inf(fnum,infci);
      for i=1:fnum if c(i)==Inf in(i,infci)=0; else in(i,1:c(i))=1./([1:c(i)]*a(i,1)); endif endfor
      %display(in);
  endif
  if exist('tbs') toin=zeros(fnum,1); tbcin=1; else tbcin=0; endif %initialise first token arrivals
  out=[Inf(fnum,1)]; t=0;
  %note: first arrival to each queue at t=0 (transient phase elliminates correlation)

%state and inter-event monitors
  X=zeros(fnum,1); if tbcin Xt=zeros(fnum,1); endif
  timeX=zeros(fnum,min(max(s),smax)+1);
  S=[0;0;0]; %currently served flows / time in service / occupied servers - 0 required for min/max etc.
  fi=0; fs=0; %current event's flow fi and next served flow fs
  ni=0; %ni = current server to accept/finish serving
  %queue monitors (lastA = time since last arrival of flow, ...)
  lastA=zeros(fnum,1); lastB=zeros(fnum,1); lastD=zeros(fnum,1);
  simT=0;
  %collectors for doing the statistics
  sumX=zeros(fnum,1); sumQ=zeros(fnum,1);
  sumA=zeros(fnum,1); sumB=zeros(fnum,1); sumD=zeros(fnum,1);
  %counters for arrivals, departures, losses, ???, servings, ...
  i=0; j=0; k=0; o=0; r=0; w=0; rrs=[]; sV=zeros(fnum,1);
  %rows to collect inter-event/-departure/-arrival/-blocking/-waiting times
  %xi=zeros(fnum, 2*loop);
```

```
  xi={[0]}; for fin=2:fnum xi={xi{},[0]}; endfor %initialise cell-array xi(fi) of num-arrays xi(fi){}
  txi=zeros(fnum,1); %per flow time spent in state xi
  di=zeros(fnum, loop); ai=zeros(size(di)); bi=zeros(size(di)); wi=zeros(size(di)); si=zeros(size(di));
  xistartindex=0; xiendindex=Inf;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%   main loop begin   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%event by event: update sim-time, generate next event, handle current event, loop until no events remain
  monitor=0; %esbusy=0;
  do
    tinext=min(in(:)); tonext=min(out(:)); t=min(tinext,tonext);
    if (t<0 || t==Inf) %error('test error: next event time =%f',t); endif
      printf('test error: next event time =%f \n',t); keyboard; endif

    %%%%%%%%%%%% do token arrivals %%%%%%%%%%%%%
    while tbcin && min(toin(:))<t %do earlier arriving tokens first
      toat=min(toin(:)); [fi,ci]=find(toin==toat); fi=fi(1); ci=ci(1); %get token time and indices
      if Xt(fi)<tbs(fi) Xt(fi)++; endif %allow token to enter bucket (nothing to be done if not)
      %schedule next token arrival
      toin(fi,1)=toat+feval('distrGen',deblank(Serv(fi,:)),1/tar(fi),num2cell(b(fi,2:end)){:});
      %schedule departure using the just arrived token
      if (X(fi)-sV(fi)>0 && sum(sV)<n)  %should never enter this for infinite servers or processor sharing
        if (Xt(fi)>sV(fi)) %if token available
          if (ps || n==inf) error('sim runtime error: should not have entered this departure case'); endif
          nfi=fi;   %nfi: index of next flow to serve, initialised with index of arrived token
          if (~es || n-sum(sV)==1) %enter here also with es if only one server is available
            out=[out,Inf(fnum,1)]; ni=columns(out);
            out(nfi,ni)=toat+feval('distrGen',deblank(Serv(nfi,:)),1/b(nfi,1),num2cell(b(nfi,2:end)){:});
            S=[S,[nfi;-toat;1]]; twi=0; %add new client to server list
            sV(nfi)+=1; %esbusy+=1;
            %display([nfi,X(nfi),sV(nfi)]);
            %out(nfi)=feval(Serv,num2cell(b(nfi,:)){:});
            switch deblank(pol(2,:)) %-> here queueing comes in - remove scheduled client from Waiting
              case 'FIFO'; twi=Waiting(nfi){:}(1); Waiting(nfi)=Waiting(nfi){:}(2:end);
              case 'LIFO'; twi=Waiting(nfi){:}(end); Waiting(nfi)=Waiting(nfi){:}(1:end-1);
              case 'RAND'; nci=ceil(length(Waiting(nfi){:})*rand(1));
                twi=Waiting(nfi){:}(nci); Waiting(nfi)=[Waiting(nfi){:}(1:nci-1),Waiting(nfi){:}(nci+1:end)];
              otherwise error('\n ERROR: queueing policy \"%s\" is not known \n',pol(2,:)); return;
            endswitch
            if monitor w++; wi(nfi,w)=twi+toat; endif %monitor waiting time (fi or nfi ???)
          else esj=((X.-sV)>0); qix=sum(esj); %qix queues with X(i)-sV(i)>0 waiting customers each
            warning('THIS part on ES serving with Token Buckets is not finished yet!');
            while qix>0 % TO BE DONE - if poissible at all... serve all populated queues equally
              if esj(nfi)>0 esk=floor((n-sum(sV))/qix+rand); while esk>(n-sum(sV)) esk--; endwhile
                if esk>0
                  out=[out,Inf(fnum,1)]; ni=columns(out);
                  out(nfi,ni)=toat+feval('distrGen',deblank(Serv(nfi,:)),1/(esk*b(fi,1)),...
                    num2cell(b(nfi,2:end)){:});
                  S=[S,[nfi;0;esk]]; twi=0; %add new client to server list
                  sV(nfi)+=esk; %esbusy+=esk;
                  %printf('\n'); display(S); display(sV');
                  switch deblank(pol(2,:)) %-> here queueing comes in - remove scheduled client from Waiting
                    case 'FIFO'; twi=Waiting(nfi){:}(1); Waiting(nfi)=Waiting(nfi){:}(2:end);
                    case 'LIFO'; twi=Waiting(nfi){:}(end); Waiting(nfi)=Waiting(nfi){:}(1:end-1);
                    case 'RAND'; nci=ceil(length(Waiting(nfi){:})*rand(1));
                      twi=Waiting(nfi){:}(nci); Waiting(nfi)=[Waiting(nfi){:}(1:nci-1),...
                        Waiting(nfi){:}(nci+1:end)];
                    otherwise error('\n ERROR: queueing policy \"%s\" is not known \n',pol(2,:)); return;
                  endswitch
                  if monitor w++; wi(nfi,w)=twi+toat; endif %monitor waiting time
                endif
                qix--;
              endif
              r=mod(++r,fnum); nfi=r+1;
            endwhile
          endif
          tonext=min(out(:)); t=min(tinext,tonext); %there may be a new, earlier departure scheduled now
        endif
      endif
      %dotoarr=(min(toin(:))<=t); %check if more tokens arrive before the next arrival or departure occurs
    endwhile
```

```
  %%%%%%%%%%%% end token arrival %%%%%%%%%%%%

  if (t<0 || t==Inf) %error('test error: next event time =%f',t); endif
    printf('test error: next event time =%f \n',t); keyboard; endif
  if (min(in(:))<Inf) %get flow and customer index of next arrival
    [fi,ci]=find(in==tinext); fi=fi(1); ci=ci(1);
  else fi=0; ci=0;
  endif
  if (min(out(:))<Inf) %get flow and server index of next departure
    [fs,ni]=find(out==tonext); fs=fs(1); ni=ni(1);
  else fs=0; ni=0;
    if find(S(1,:)) display(out); fprintf('\n Currently served flows: ');
      fprintf('%2d/%3.2f/%d ',S(:,:));
      %for sci=1:length(S(1,:)) fprintf('%2d/%3.2f ',S(1,sci),S(2,sci)); endfor
      error('runtime error -> no departures scheduled but Service-array is not emptied!');
    endif
  endif
  if (tinext<=tonext) %find index of a currently idle server, zero if non is idle
    if max(S(1,:))==0 ni=1; elseif min(S(1,2:end))>0 ni=0; else ni=find(S(1,:)==0)(1); endif
  else fi=fs; endif %in case of departure we use the server that becomes idle to server next waiting client
  in.-= t;                    %subtract time since last event from scheduled arrival events
  if tbcin toin.-= t; endif   %subtract time since last event from scheduled token arrival events
  out.-= t;                   %subtract time since last event from scheduled service completions
  S(2,:).+=t;                 %add inter-event time to times in service (for service time monitor)
  sV=zeros(fnum,1);           %initialise vector indicating how often a queue(flow) is currently served
  if ~es for sci=2:columns(S) sV(S(1,sci))+=1; endfor
  else for sci=2:columns(S) sV(S(1,sci))+=S(3,sci); endfor
  endif
   %display([X',sV',S(1,:)]);
   %fprintf('\n t=%f, fi=%d ',t,fi);
  for fid=1:fnum
    %xi{fid}(end)+=t*X(fid);  %accumulate state holding period, for current fi new is added at end of loop
    Waiting(fid)=Waiting(fid){:}+t; %add inter-event time to waiting times
  endfor
  if monitor %collect infos
    simT+=t;  %sum up simulation time (duration of simulation since monitoring start)
    lastA.+=t;    %add inter-event time to times since last arrival
    lastD.+=t;    %add inter-event time to times since last departure
    lastB.+=t;    %add inter-event time to times since last blocking
    %collect system states wieghted by the time they last (sec) normalised to the service rate (1/sec)
    %xi(:,i+j-trans)=t*X; %index is current arrivals+departures shifted up trans leaving trans till 2*loop
    %accumulate state holding period, for current fi new is added post event handling
    txi.+=t; xi{fi}(end)+=txi(fi)*X(fi); txi(fi)=0;
    %Q(num2cell([X'+1 fs+1]){:})+=t; %add inter-event time to time spent in current state
    timeX(:,min(X,smax)+1)+=t;    %add inter-event time to time spent in current state
    sumX(:)+=t*X(:);              %sum time weighted system filling
    sumQ(:)+=t*(X(:).-sV(:));     %sum queue filling (no difference for leave-after-service regime)
  endif
  if (t<0 || t==Inf) printf('test error: next event time =%f \n',t); keyboard; endif
  prevX=sum(X);       %remember the system filling prior event handling
  prevS=columns(S)-1; %remember the server occupation prior event handling
  if (~any(prevSigm) && any(g(:)) && any(X)) prevSigm=g(:,1)./sum(g(:,1).*X); %init first sharing factor
  elseif (~any(prevSigm) && any(g(:))) prevSigm=ones(fnum,1); %init weighted fair sharing prevSigm
  elseif (~any(prevSigm) && any(X)) prevSigm=ones(fnum,1)./prevS; %unweighted fair sharing
  elseif (~any(prevSigm)) prevSigm=ones(fnum,1); %server is idle and no wieghting (factor=1)
  endif
  if (ps && prevS != prevX) error('impossible error'); endif
if (tinext<=tonext)

  %fprintf('arrival%d ',fi);    %         ********* ARRIVAL *********

  if exist('Bat') bats=round(feval('distrGen',deblank(Bat(fi,:)),1,num2cell(a(fi,2:end)){:}));
    if i<trans bats=min(bats,trans-i); elseif i==trans bats=1; else bats=min(bats,loop-i); endif
    %required for monitoring control
  else bats=1; endif
  for ban=1:bats
    if monitor ai(fi,i-trans)=max(lastA(fi),eps); sumA(fi)+=lastA(fi); endif  %collect/sum inter-arrival times
    lastA(fi)=0;  %add new inter-arrival time and reset time since last arrival
    if isscalar(s) letin=sum(X)<s;
    elseif (exist('rdrop') && comqu) letin=1-rdrop(fi,sum(X)+1); %random drop
      if letin<1 && letin>0 letin=rand()<letin; endif %letin=rand()>rdrop(fi,sum(X)+1);
    elseif comqu letin=sum(X)<s(fi);
```

```
    else letin=X(fi)<s(fi); endif %entry allowed?
    if rows(s)==1 && ~comqu error('comqu test error'); endif
    if ~isscalar(letin) error('sim runtime error <- entry evaluation yields a non-scalar result'); endif
    if letin %(comqu && sum(X(:))<s) || (~comqu && X(fi)<s(fi))   %check space in queue(s)
      X(fi)++;    %customer enters the system -> adjust system state
      if (~tbcin || Xt(fi)>sum(sV)) && (sum(sV)<n || ps) % || sum(X)<=n)
        %immediately schedule depature (at least one server is idle)
        out=[out,Inf(fnum,1)]; ni=columns(out);
        %set inital holding time mht
        if prevSigm(fi)>1 error('impossible service share %f for flow %d',prevSigm(fi),fi); endif
        if ps mht=1/(prevSigm(fi)*b(fi,1)); esk=1; % note: re-adjusted to current shares at the end!
        elseif es esk=n-sum(sV); %esk
          mht=1/(esk*b(fi,1)); %esbusy+=esk; %note: one served -> makes all servers busy
        else mht=1/b(fi,1); esk=1; endif
        if mht<b(fi,1) error('impossible mean service time %f for flow %d',mht,fi); endif
        out(fi,ni)=feval('distrGen',deblank(Serv(fi,:)),mht,num2cell(b(fi,2:end)){:});
        S=[S,[fi;0;esk]]; twi=0; %add new client to server list
        sV(fi)=esk;  %display(S); display(sV');
        %monitor zero waiting time
        if monitor w++; wi(fi,w)=eps; endif %use eps so delZeros does not remove entries
      else %add current arrival to waiting customers
        if (ps || n==inf) error('sim runtime error <- should not have entered waiting case'); endif
        Waiting(fi)=[Waiting(fi){:},0];
      endif
      if (cm(fi)<Inf) %Engset case: arrival accepted -> generate next after service!
        ci=find(in(fi,:)<=eps)(1);
        if ci==infci error('delayed scheduling for infinite population index not foreseen!'); endif
        darr=[darr,fi]; in(fi,ci)=Inf;
      endif
    else %blocked arrival
      if (s(min(rows(s),fi))==Inf) error('sim runtime error: should not have entered blocking case'); endif
      if monitor o++; bi(fi,o)=lastB(fi); sumB(fi)+=lastB(fi); endif  %collect/sum inter-blocking times
      lastB(fi)=0;    %reset time since blocking monitor
      if (i<loop-length(find(in<Inf)) && cm(fi)<Inf)
        %Engset case: immediately schedule next arrival if current was blocked
        if isscalar(c) cs=max(cm-sum(X),0); else cs=max(cm-X,0); endif
        ci=find(in(fi,:)<=eps)(1);
        if ci==infci error('delayed scheduling for infinite population index not foreseen!'); endif
        in(fi,ci) = feval('distrGen',deblank(Arr(fi,:)),1/a(fi,1),num2cell(a(fi,2:end)){:});
        %if (ci && in(fi,ci)<=0) error('next arrival -- negative time must not result!'); endif
      else in(fi,ci)=Inf;
      endif
    endif
    i++; % count the arrival (accepted or blocked)
  endfor
    if (i<=loop-length(find(in<Inf)) && cm(fi)==Inf)
      %Erlang case: schedule next arrival after arrival of previouse
      in(fi,infci)=feval('distrGen',deblank(Arr(fi,:)),1/a(fi,1),num2cell(a(fi,2:end)){:});
      %if (in(fi,infci)<=0) error('next arrival -- negative time must not result!'); endif
    else in(fi,infci)=Inf; endif
else

  %fprintf('departure%d ',fi);    %       ********* DEPARTURE *********

  if tinext<=tonext error('impossible error'); endif
  if (X(fi)<=0)
    error('\n sim runtime error: departure from flow %d at X(%d)=%d occured! \n',fi,fi,X(fi)); break; endif
  j++; X(fi)--; %adjust system state
  if tbcin Xt(fi)--; if Xt(fi)<0 keyboard();
    error('sim runtime error: token bucket depleted below zero Xt(%d)=%d',fi,Xt(fi)); endif, endif
  %display([X',sV']);
  if (sV(fi)<0) error('\n sim runtime error: service vector entry for flow %d became %d! \n',{}
    fi,sV(fi)); break; endif
  ni=find(out(fi,:)<eps)(1); %index of departure in the out-matrix, first found if multiple departure
  nsi=ni; %nsi=find(S(1,:)==fi); %indices of clients from same flow in server list - which finishes now?
  sV(fi)-=S(3,nsi); %esbusy-=S(3,nsi);
  lastS=S(2,nsi); %current departure's time in service
  out=[out(:,1:ni-1),out(:,ni+1:end)]; S=[S(:,1:nsi-1),S(:,nsi+1:end)];   % remove served customer
  if monitor k++; di(fi,k)=lastD(fi); sumD(fi)+=lastD(fi); si(fi,k)=lastS; endif %monitors
  lastD(fi)=0;    %reset time since last departure monitor
  if (max(X.-sV)>0)   %should never enter this for infinite servers or processor sharing
    if (~tbcin || Xt(fi)>sum(sV)) %if token required & available
```

```
    if (ps || n==inf) error('sim runtime error: should not have entered this departure case'); endif
    [xmax,nfi]=max(X);  %nfi: index of next flow to serve, initialised with most filled queue (xmax)
    switch deblank(pol(1,:)) %-> here scheduling comes in :-)   %select flow to serve next
      case 'SQ'; wlist=NaN(1,fnum);
      %SingleQueue / FCFS (use only with multiple services, else maxQ or RR are best)
        for r=1:fnum if isempty(Waiting(r){:}) wlist(r)=-1; else wlist(r)=Waiting(r){:}(1); endif, endfor,
        [wmax,nfi]=max(wlist); %display([i,wlist,nfi]); %display(Waiting); % serve the longest waiting
        if wlist(nfi)<0 error('sim runtime error: queue with no waiting client should be served'); endif
      case 'SP';
        if any(g) tempg=g(:); do nfi=find(g==max(tempg))(1); tempg(nfi)=NaN; until X(nfi)-sV(nfi)>0
          %StrictPriority by weight
        else nfi=1; while X(nfi)-sV(nfi)<=eps nfi++; endwhile endif %StrictPriority by lower index
        %else for nfi=1:fnum if X(nfi)-sV(nfi)>0 break; endif endfor endif %StrictPriority by index
      case 'RQ'; r=ceil(sum(X-sV>0)*rand()); nfi=find(X-sV>0)(r); %RandomQueue (seems ok, not approved)
      case 'RR'; do r=mod(++r,fnum); nfi=r+1; until (X(nfi)-sV(nfi)>0) %RoundRobin
      case 'WRQ'; gr=cumsum((g(:).*((X(:).-sV(:))>0))./(g(:)'*((X(:).-sV(:))>0))); %WeightedRandomQueue
        r=rand(); nfi=1; while (X(nfi)-sV(nfi)<=eps || gr(nfi)<=r) nfi++; endwhile
      case 'WRQ+'; gr=cumsum(g(:).*(X(:).-sV(:)))./(g(:)'*(X(:).-sV(:))); %WaitingWeightedRandomQueue
        r=rand(); nfi=1; while (X(nfi)-sV(nfi)<=eps || gr(nfi)<=r) nfi++; endwhile
      case 'WRR'; if ~any(rrs) for r=0:max(g)-1 gr=find(g.-r>0); rrs=[rrs;gr]; endfor, r=0; endif
        do r=mod(++r,length(rrs)); nfi=rrs(r+1); until (X(nfi)-sV(nfi)>0) %WeightedRoundRobin
      case 'maxQ'; [xmax,nfi]=max(X.-sV); %most filled queue without already served clients
      case 'minQ'; for qix=1:fnum %least filled queue first
        if (X(qix)-sV(qix)>0) && (X(qix)-sV(qix)<(X(nfi)-sV(nfi))) nfi=qix; endif endfor
      otherwise error('\n ERROR: scheduling policy \"%s\" is not known \n',pol(1,:)); return;
    endswitch
    %schedule next departure
    if (~es || n-sum(sV)==1) %enter here also with es if only one server is available
      out=[out,Inf(fnum,1)]; ni=columns(out);
      out(nfi,ni)=feval('distrGen',deblank(Serv(nfi,:)),1/b(nfi,1),num2cell(b(nfi,2:end)){:});
      S=[S,[nfi;0;1]]; twi=0; %add new client to server list
      sV(nfi)+=1; %esbusy+=1;
      %display([nfi,X(nfi),sV(nfi)]);
      %out(nfi)=feval(Serv,num2cell(b(nfi,:)){:});
      switch deblank(pol(2,:)) %-> here queueing comes in - remove scheduled client from Waiting
        case 'FIFO'; twi=Waiting(nfi){:}(1); Waiting(nfi)=Waiting(nfi){:}(2:end);
        case 'LIFO'; twi=Waiting(nfi){:}(end); Waiting(nfi)=Waiting(nfi){:}(1:end-1);
        case 'RAND'; nci=ceil(length(Waiting(nfi){:})*rand(1));
          twi=Waiting(nfi){:}(nci); Waiting(nfi)=[Waiting(nfi){:}(1:nci-1),Waiting(nfi){:}(nci+1:end)];
        otherwise error('\n ERROR: queueing policy \"%s\" is not known \n',pol(2,:)); return;
      endswitch
      if monitor w++; wi(nfi,w)=twi; endif %monitor waiting time (fi or nfi ???)
    else esj=((X.-sV)>0); qix=sum(esj); %qix queues with X(i)-sV(i)>0 waiting customers each
      %warning('THIS part on ES serving is not finished yet!');
      while qix>0 % TO BE DONE - if poissible at all... serve all populated queues equally
        if esj(nfi)>0 esk=floor((n-sum(sV))/qix+rand); while esk>(n-sum(sV)) esk--; endwhile
          if esk>0
            out=[out,Inf(fnum,1)]; ni=columns(out);
            out(nfi,ni)=feval('distrGen',deblank(Serv(nfi,:)),1/(esk*b(fi,1)),num2cell(b(nfi,2:end)){:});
            S=[S,[nfi;0;esk]]; twi=0; %add new client to server list
            sV(nfi)+=esk; %esbusy+=esk;
            %printf('\n'); display(S); display(sV');
            switch deblank(pol(2,:)) %-> here queueing comes in - remove scheduled client from Waiting
              case 'FIFO'; twi=Waiting(nfi){:}(1); Waiting(nfi)=Waiting(nfi){:}(2:end);
              case 'LIFO'; twi=Waiting(nfi){:}(end); Waiting(nfi)=Waiting(nfi){:}(1:end-1);
              case 'RAND'; nci=ceil(length(Waiting(nfi){:})*rand(1)); twi=Waiting(nfi){:}(nci);
                Waiting(nfi)=[Waiting(nfi){:}(1:nci-1),Waiting(nfi){:}(nci+1:end)];
              otherwise error('\n ERROR: queueing policy \"%s\" is not known \n',pol(2,:)); return;
            endswitch
            if monitor w++; wi(nfi,w)=twi; endif %monitor waiting time
          endif
          qix--;
        endif
        r=mod(++r,fnum); nfi=r+1;
      endwhile
    endif
  endif
endif
if (i<loop-length(find(in<Inf)) && cm(fi)<Inf && any(darr==fi))
  %Engset case: schedule next arrival after service of previouse
  daind=find(darr==fi)(1);
  if isscalar(c) cs=max(cm-sum(X),0); else cs=max(cm-X,0); endif
```

```
      ci=find(in(fi,:)==Inf)(1);
      if ci==infci error('delayed scheduling for infinite population not foreseen!'); endif
      in(fi,ci)=feval('distrGen',deblank(Arr(fi,:)),1/a(fi,1),num2cell(a(fi,2:end)){:});
      darr=[darr(1:daind-1),darr(daind+1:end)]; %remove the now scheduled flow
    endif
    if (ci && in(fi,ci)==0) error('next arrival -- zero must not result!'); endif
  endif
    %                         ********* a posteriori event handling *********

    %if sum(X)<=0 esbusy=0; endif %dirty patch to problem with esbusy
    if (ps && sum(X)>0) %adjust out-times for processor sharing
      %if any([prevX,sum(X)]<eps) error('test error'); endif
      if any(g(:)) savSigm=prevSigm; %priority weighted processor sharing
        sigm=min(1,g(:,1)./sum(g(:,1).*X(:))); %calculate current server shares (<=1)
        toscale=prevSigm./sigm; %to be checked! (1+sigm.^(-1))./(1+prevSigm.^(-1));
        prevSigm=sigm;
        %printf(' -> contiune:dbcont, quit:dbquit <- \n'); fflush(stdout); keyboard();
      elseif (prevX>0) toscale=sum(X)/prevX; %egalitarian processor sharing
        prevSigm.*=toscale.^(-1); error('test error'); %??? to be checked !!!
      endif
      if (any(toscale==Inf) || any(toscale<=0))
        error('sim runtime error: infinite or negative scaling may not happen!'); endif
      % adjust out-times to consider service rate change in case of processor sharing
      savOut=out;
      for pfi=1:fnum if toscale(pfi)<Inf out(pfi,:).*=toscale(pfi); endif endfor %rescale departure times
    elseif (ps && any(g(:))) savSigm=prevSigm; %reset <-- server became idle
      prevSigm=ones(fnum,1); sigm=prevSigm; toscale=sigm;
      if columns(out)>1 display(out);
        error('sim runtime error: system idle but [out] was not cleared!'); endif
    endif
    % monitoring control
    if (i==trans) xistartindex=max(xistartindex,i+j-trans); %remember index prior monitoring start
    elseif (i==trans+1) monitor=1;                          %start monitoring
    elseif (i==loop) monitor=0;                             %end monitoring
      xiendindex=min(xiendindex,i+j-trans);                 %remember index post monitoring end
    endif
    %                         ********* report simulation progress *********

    if (i<loop && tinext<=tonext && rem(i-trans,1000)==0) %  && tinext<=tonext (at arrivals only)
      fprintf('\n%3.1f%% at %5.4f done: i=%d, j=%d, k=%d, o=%d; \n',(i-trans)/(loop-trans)*100,l,i,j,k,o);
      fprintf(' Flow \t nextArr \t [inQ] \t nextDep \n');
      for fci=1:fnum fprintf(' %d: \t',fci);
        if (cm(fci)==Inf) fprintf(' %5.2f  ',in(fci,end));
        elseif length(in(fci,:))>5 fprintf(' %5.2f(%d)', min(in(fci,:)), length(find(in(fci,:)<Inf)));
        else fprintf(' %4.2f',in(fci,1:end-1));
        endif, fprintf('\t [%d]  \t',X(fci));
        if length(out(fci,:))>5 fprintf(' %5.2f(%d)', min(out(fci,:)), length(find(out(fci,:)<Inf)));
        else fprintf(' %4.2f',out(fci,2:end));
        endif, fprintf('\n');
      endfor
      fprintf(' Server occupation:\t [%d] comp. of:',length(S(1,2:end)));
      if length(S(1,:))>5 for fci=1:fnum
        fprintf(' %d(%d)%3.2f/%d',fci,length(find(S(1,2:end)==fci)),max(S(2,S(1,:)==fci)),sum(S(3,S(1,:)==fci)));
        endfor
      else fprintf('%2d/%3.2f/%d ',S(:,2:end));
      endif, fprintf('\n');
      if any(darr) fprintf(' Pending arrivals: '); fprintf('%2d ',length(darr)); fprintf('\n'); endif
      if (ps) printf(' service shares: '); printf('%5.4f ',X.*prevSigm); printf('\n'); endif %&& prevX>0
      %if (ps) printf(' out-scaling {%d}: ',fi); printf('%5.4f ',toscale(:)); printf('\n'); endif %&& prevX>0
      %if (any(sigm) && sum(X)>0) display([sigm'.^(-1);savSigm'.^(-1);(savSigm./sigm)']); endif
      %printf(' / '); printf('%5.4f ',savSigm(:)); printf(' - '); printf('%5.4f ',sigm(:)); endif
      %display(prevOut);
      fflush(stdout);
      %keyboard(); %debugging -> continue: dbcont, quit: dbquit (=exit)
    endif
    if monitor xi(fi)=[xi(fi){},0]; endif %add new entry to flow's state monitor
  until (min(in)==Inf && min(out)==Inf)
  %end loop if no arrivals or departures are left
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  %%%   main loop end   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%final progress report
```

```
  fprintf('\n%3.1f%% at %5.4f done: i=%d, j=%d, k=%d, o=%d; \n',i/loop*100,l,i,j,k,o);
  display([in X out]);  display([S(:)' simT]);
  if (i!=loop) warning('\n generated arrival events (%d) != loop (%d) \n',i,loop); endif

%evaluate results using recorded traces %%%
  %shrink and normalise xi to actual number of monitored events and time passed
  %[trs,tri]=min(sum(xi,2)); if trs==0 error('empty xi trace for flow %d should not happen!',tri); endif
  for fid=1:fnum
    xi{fid}.*=length(xi{fid})/simT; %normalise by total sim-time
    trs=sum(xi{fid}); if trs==0 error('empty xi trace for flow %d should not happen!',fid); endif
  endfor
  if ~(o>0) bi=zeros(fnum,1); cvB=NaN(fnum,1); ebB=NaN(fnum,1); endif
  for qfi=1:fnum
    aq=delZeros(ai(qfi,:)); aq(find(aq==eps))=0;
    mA(qfi)=1/mean(aq); cvA(qfi)=std(aq)*mA(qfi); ebA(qfi)=confid(aq,95);
    el(qfi)=mA(qfi);    %current load estimated from arrival trace
    norml(qfi)=mA(qfi)/(loop-trans); %load dependent factor to normalise resuts if necessary
    sq=delZeros(si(qfi,:)); %may be empty in case of starvation!
    if any(sq) mS(qfi)=mean(sq); cvS(qfi)=std(sq)*mS(qfi); ebS(qfi)=confid(sq,95);
    else mS(qfi)=NaN; cvS(qfi)=NaN; ebS(qfi)=NaN; endif %??? confid*mean NO !!!
    %printf('mean Ts=%f from %d samples \n',mean(si(qfi,si(qfi,:)>0)),length(find(si(qfi,:)>0)));
    dq=delZeros(di(qfi,:)); %may be empty in case of starvation!
    if any(dq) mD(qfi)=1/mean(dq); cvD(qfi)=std(dq)*mD(qfi); ebD(qfi)=confid(dq,95);
    else mD(qfi)=NaN; cvD(qfi)=NaN; ebD(qfi)=NaN; endif
    %dnorml(qfi)=sum(dq)/k; %service dependent factor to normalise resuts to service time
    bq=delZeros(bi(qfi,:));
    if isempty(bq) mB(qfi)=0; cvB(qfi)=NaN; ebB(qfi)=NaN;
    else mB(qfi)=1/mean(bq); cvB(qfi)=std(bq)*mB(qfi); ebB(qfi)=confid(bq,95); endif
    xnorml(qfi)=length(xi{qfi})/(columns(aq)+columns(bq)+columns(dq)); %xi is recorded for every event
    %xq=xi(qfi,:)*xnorml; %display(xnorml); %required for multi-queue SP sim ?????
    xq=xi{qfi}.*xnorml(qfi); %display(xnorml(qfi));
    mX(qfi)=mean(xq); cvX(qfi)=std(xq)/mX(qfi); ebX(qfi)=confid(xq,95);
    wq=(delZeros(wi(qfi,:)).-eps); %display(wq'); %remove eps, assumed irrelevant for any tw>0
    if isempty(wq) mW(qfi)=0; cvW(qfi)=NaN; ebW(qfi)=NaN;
    else mW(qfi)=mean(wq); %is zero if zero waiting has been monitored
      if mW(qfi)<eps cvW(qfi)=NaN; else cvW(qfi)=std(wq)/mW(qfi); endif
      ebW(qfi)=confid(wq,95); endif
    %flow time calculation
    if ps mF(qfi)=mS(qfi); cvF(qfi)=cvS(qfi); ebF(qfi)=ebS(qfi);
    else mF(qfi)=mX(qfi)/mD(qfi); cvF(qfi)=NaN; ebF(qfi)=NaN; %cv and eb calc?
    %mF(qfi)=mX(qfi)/mA(qfi); if zero waiting for blocked load shall be included
    %else mF(qfi)=mW(qfi)+mS(qfi);
    %  cvF(qfi)=2*(cvW(qfi)*mW(qfi)+cvS(qfi)*mS(qfi))/(mW(qfi)+mS(qfi)); %ok (if independent)
    %  ebF(qfi)=NaN; %calculate?
    endif
    fprintf('queue%d: load=%5.4f, Pb=%5.4f, lin=%5.4f', qfi, el(qfi), mB(qfi)/mA(qfi), mA(qfi)-mB(qfi));
    fprintf(', depnr=%d, corrf=%5.4f \n', length(find(si(qfi,:)>0)), xnorml(qfi));
    fprintf(' mX=%5.4f, cvX=%5.4f, confX=%5.4f \n', mX(qfi), cvX(qfi), ebX(qfi));
    fprintf(' mA=%5.4f, cvA=%5.4f, confA=%5.4f \n', mA(qfi), cvA(qfi), ebA(qfi));
    fprintf(' mS=%5.4f, cvS=%5.4f, confS=%5.4f \n', mS(qfi), cvS(qfi), ebS(qfi));
    fprintf(' mD=%5.4f, cvD=%5.4f, confD=%5.4f \n', mD(qfi), cvD(qfi), ebD(qfi));
    fprintf(' mB=%5.4f, cvB=%5.4f, confB=%5.4f \n', mB(qfi), cvB(qfi), ebB(qfi));
    fprintf(' mW=%5.4f, cvW=%5.4f, confW=%5.4f \n', mW(qfi), cvW(qfi), ebW(qfi));
    fprintf(' mF=%5.4f, cvF=%5.4f, confF=%5.4f \n', mF(qfi), cvF(qfi), ebF(qfi));
    fflush(stdout);
  endfor
  fprintf('\n');
  %keyboard(); %debugging: continue: dbcont, quit: dbquit (=exit)

%print histogram of some monitored distribution(s) -- adjust source on demand!
  if draw
    tmax=histp(1); bins=histp(2);
    %%adjust record sizes - not required with current HistPlot
    %slength=min(length(aq),length(dq)); aq=aq(1:slength); dq=dq(1:slength);
    %figure(1); hold("on"); HistPlot(aq,tmax,bins,0); HistPlot(dq,tmax,bins,0); hold("off");  %plot pdfs
    %figure(2); hold("on"); cHistPlot(aq,tmax,bins,0); cHistPlot(dq,tmax,bins,0); hold("off");  %plot cdfs
    nh=~ishold; %samp,tail,bins,lplot,plcolor
    figure(1); if nh hold("on"); endif; HistPlot(dq,tmax,bins,0); if nh hold("off"); endif %plot pdfs
    figure(2); if nh hold("on"); endif; cHistPlot(dq,tmax,bins,0); if nh hold("off"); endif %plot cdfs
    figure(1); %dummy command to actually plot figure 2
    %%waiting time distribution
    %wnorml=simT/(loop-trans);
```

```
    %cwq=delZeros(wq); %conditional waiting times: without zero-waiting
    %figure(1); hold("on"); HistPlot(cwq,tmax,bins,0); hold("off");  %plot pdfs
    %figure(2); hold("on"); cHistPlot(wq,tmax,bins,0); cHistPlot(cwq,tmax,bins,0,2); hold("off");  %plot cdfs
  endif

%collect trace based performance results in return record (statis)
  statis=NaN(fnum+1,19);
  if (n<Inf && all(c<Inf)) statis(1:fnum,1)=(c(:).*a(:,1))./(1+a(:,1)./(n*b(:,1))); %effectively offered load
  elseif (n<Inf && any(c<inf)) warning('c=[c1,c2,inf,etc.]-case effective offered load calc. to be done');
  elseif (n<Inf) statis(1:fnum,1)=a(:,1)./(n*b(:,1)); %load per flow rho_i=a_i/(n*b_i)
  else statis(1:fnum,1)=a(:,1)./b(:,1); endif
  %display([a,b,statis(1:fnum,1)])
  statis(fnum+1,1)=sum(statis(1:fnum,1)); %system load =sum_i{rho_i}
  statis(1:fnum,2:10)=[mA(:),ebA(:),cvA(:),mD(:),ebD(:),cvD(:),mB(:),ebB(:),cvB(:)]; %flow statistics
  %% the monitored mW is here overwritten by calculated value
  if any(mW(:)==NaN) for fid=1:fnum
    if (mW(fid)==NaN && mS(fid)!=NaN && mF(fid)!=NaN) mW(fid)=mF(fid)-mS(fid); endif endfor endif
  mQ=mD(:).*mW(:); %using Little with carried load (not the offered load)
  ebQ=abs(mD(:).*ebW(:)); %error-bar values change linearly with linear scaling???
  statis(1:fnum,11:19)=[mX(:),ebX(:),mQ(:),ebQ(:),mF(:),ebF(:),mW(:),ebW(:),cvW(:)]; %performance statistics
  %last line used for total system results
  statis(fnum+1,2)=sum(statis(1:fnum,2)); %total arrival rate
  statis(fnum+1,5)=sum(statis(~isnan(statis(1:fnum,5)),5)); %total departure rate
  statis(fnum+1,8)=sum(statis(~isnan(statis(1:fnum,8)),8)); %total blocking rate
  statis(fnum+1,11)=sum(statis(1:fnum,11)); %total system filling
  statis(fnum+1,13)=sum(statis(1:fnum,13)); %total queues filling
  dshares=statis(~isnan(statis(1:fnum,5)),5)./statis(fnum+1,5); %departure shares
  statis(fnum+1,15)=statis(~isnan(statis(1:fnum,5)),15)' * dshares; %averaged flow time
  statis(fnum+1,17)=statis(~isnan(statis(1:fnum,5)),17)' * dshares; %averaged waiting time

%point simulation finished
if pageStdOut more('on'); endif %reset environment to previouse behaviour
endfunction
```

A.I.3 Filling and solving the Q-matrix for finite multi-flow systems

Here we briefly present how the *Q-matrix* for finite systems can be set-up in *octave* for *any number of flows*. As example we choose the *weighted random queue* policy (WRQ) presented in section 4.1.3, figure 4.21 and 4.22.

Setting up the finite Q-matrix for WRQ policy

```
if isscalar(s) coqu=true; sq=s*ones(1,fn); else coqu=false; sq=s; endif  %queue-size(s) s, share=coqu
Qvec=cell2mat(nthargout([1:fn+1],@ind2sub,[(sq+1)(:)',fn],(1:fn*prod(sq+1))'));  %fn+1 dim Q-indices
Qvec.-=1; Qvec(2:end,end).+=1; %convert matrix indices to state indices (i,j,k,q)
if coqu Qvec(find(sum(Qvec,2)-Qvec(:,end)>s),:)=[]; endif %remove upper triangle
for j=1:fn Qvec(find(Qvec(:,j)==0 & Qvec(:,end)==j),:)=[]; endfor %remove impossible serving states
Vsize=rows(Qvec); %number of actually required states

for r=1:steps %load level by load level
    ar=ri.*rho(r); %arrival rates per share (ri=splitting vector, rho(r)=current load)
    Q=sparse(Vsize,Vsize); %define sparse Q matrix

    for i=1:Vsize    %state by state
        qfill=Qvec(i,1:end-1); servq=Qvec(i,end); %the state's queue filling and served queue
        if servq==0 %arrivals out of idle state
            for j=1:fn
                qfill2=qfill; qfill2(j)++; %destination queue filling
                i2=find(prod(Qvec==ones(Vsize,1)*[qfill2,j],2)); %dest Q-index
                Q(1,i2)=ar(j); %insert arrival transition rate
            endfor,
        else %insert all other transitions
            if coqu && sum(qfill)<s %common queue case
                for j=1:fn
                    qfill2=qfill; qfill2(j)++; %destination queue filling
                    i2=find(prod(Qvec==ones(Vsize,1)*[qfill2,servq],2)); %dest Q-index
                    Q(i,i2)=ar(j); %insert arrival transition rate
                endfor
```

```
            elseif ~coqu %individual queues case
                for j=1:fn
                    if qfill(j)>=sq(j) continue, endif
                    qfill2=qfill; qfill2(j)++; %destination queue filling
                    i2=find(prod(Qvec==ones(Vsize,1)*[qfill2,servq],2)); %dest Q-index
                    Q(i,i2)=ar(j); %insert arrival transition rate
                endfor
            endif
            %do departures
            qfill2=qfill; qfill2(servq)--; %queue filling after departure
            if sum(qfill2)==0 Q(i,1)=br(servq); %departure to idle state (no split possible)
            else
                qi=gi.*(qfill2>0)./sum(gi.*(qfill2>0)); %state dependent departure split
                qbr=qi.*br(servq); %aliquote split service rates
                for j=1:fn
                    if qi(j)>0 %transition exists
                        i2=find(prod(Qvec==ones(Vsize,1)*[qfill2,j],2)); %dest Q-index
                        Q(i,i2)=qbr(j); %insert departure transition rate
                    endif
                endfor
            endif, %display([qfill,indics(end);qfill2,indic2(end)]);
        endif,
    endfor,  %display(Q); %keyboard();

    %identify empty rows -> reactivate if singular matrix occurs
    %delrs=find(sum(Q,2)==0); %find all zero rows
    %if ~isempty(delrs) display(Q(delrs,:)); warning('empty rows happened!');
    %Q(delrs,:)=[]; Q(:,delrs)=[]; endif %remove idle rows and associated columns
    Q+=diag(-sum(Q,2)); %insert diagonal elements (neg row sum)
    if any(sum(Q,2)>10*eps) display([Q,sum(Q,2)]); error('faulty system Q-matrix occurred'); endif
    bvec=zeros(rows(Q),1); Q(:,end)=1; bvec(end)=1; %adjust to integrate normalization
    p=Q'\bvec; %solve equilibrium equations
    if any(p<10*eps || abs(sum(p)-1)>10*eps) p, error('faulty probability vector resulted'); endif

    for i=1:fn  %calculate and record system properties at load rho(r)
        x(r,i)=sum(Qvec(:,i).*p); %individual queue fillings
        w(r,i)=sum((Qvec(:,i)-(Qvec(:,end)==i)).*p); %waiting packets numbers
        if coqu bp(r,i)=sum(p(find(sum(Qvec,2)==s+Qvec(:,end)))); %blocking probabilities
        else bp(r,i)=sum(p(find(Qvec(:,i)==sq(i)))); endif %Pb for individual queues
        cr(r,i)=bp(r,i)*ar(i); %blocking rates
        dr(r,i)=sum(p(find(Qvec(:,end)==i)))*br(i); %throughputs
    endfor
    tf(r,:)=x(r,:)/dr(r,:); %flow times (using Little's law)
    tw(r,:)=w(r,:)/dr(r,:); %waiting times (using Little's law)

endfor
```

The input parameters required are: the number of flows `fn`, the queue size(s) `s`, where a scalar identifies a shared queue, the weights vector `gi`, the system loads vector to evaluate `rho`, the load split vector `ri`, and the service rates vector `br`. In return the code delivers: the mean values for the individual *queue fillings* `x`, the *number of waiting packets* `w`, the *blocking probabilities* `bp`, the *blocking rates* `cr`, and the *throughputs*, which equal the departure rates `dr`. From these the remaining system properties, *mean flow time* and *mean waiting time*, can be calculated using Little's law $N = \lambda T$ if we consider that here λ refers to the accepted load, which due to no leaves without service equals the departure rate.

A.II Event based network simulation

The concept presented next evolved over time and it was always a vision to realise it. Eventually, it remains a vision. Some time it may be of interest, and therefore it is included here in figure A.1. Please be aware that this is no more than a concept. If it can be realised and might be beneficial, cannot be finally answered yet.

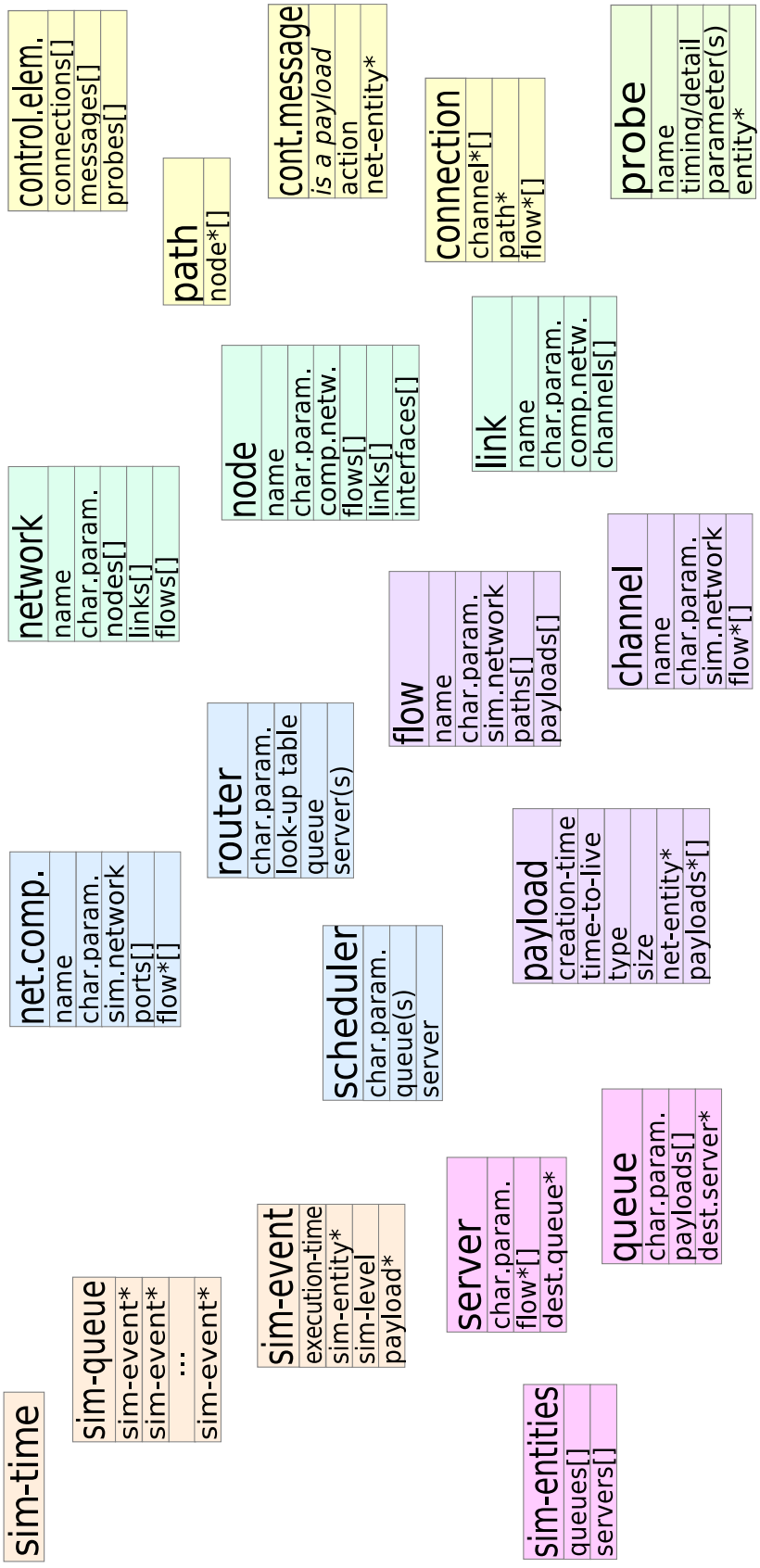


Figure A.1: Objects assumedly required for generic network simulation

Nomenclature

The used nomenclature is introduced in the course of the text where first needed. It has been intended to maximally harmonise the various nomenclatures found in the literature. However, well established but here conflicting nomenclature we did not change where advantageous for better understanding. We also tried not to introduce too many different alphabets (letter styles) and thus, some variables expressed by the same latter can have different meaning in different contexts.

This addendum lists and explains the mostly used variables and nomenclature issues applied. Note that we do not attempt to present a complete listing here. The hereafter stated definitions do not replace or overrule the in-line definitions in the course of the text, they are collected here to support a better, maybe intuitive, understanding of the mostly applied nomenclature rules.

Text styles

italic	key word, statement
slanted	product/command/item name
`typewriter`	constant/variable name, m-code

Variables and operators

$i, j, k, n, m, \ldots$	natural numbers
$a, b, c, \ldots, x, y, z$	real valued variables
t	a time based variables
τ	a duration of time
X, Y	random variable, matrix
$[a \,..\, b]$	the range of a variable
$[a_1, a_2, a_3, \ldots, a_n]$	an indexed array of a variable
$\{a, b, c, \ldots\}$	a set of independent results
$\vec{x}, y$	vectors
$\bar{x}, \overline{X}$	mean value
$\dot{x}, \frac{f(x)}{\mathrm{d}x}$	first derivation
$\hat{x}$	some expectation
$\mathrm{P}[X > x]$	the probability operator
$\mathrm{E}[X]$	the expectation operator (mean)
$Var[X]$	the variance of X
$Cov(X, Y)$	the covariance of X and Y
$F(x), f(x)$	a function of x, e.g., *cdf* and *pdf*
x^*, X^*	a variant, constraint set, or transform

Abbreviations

The provided list of abbreviations includes both abbreviations of bulky names and commonly used abbreviation like names. However, terms not identified as being an abbreviation are not included.

A	
ACK	acknowledgement
ADM	add-drop multiplexing
AS	autonomous system
ATM	asynchronous transport mode
B	
BCMP	Baskett Chandy Muntz Palacios [84]
BER	bit error rate
BGP	boarder gateway protocol
BMAP	batch Markov arrival process
C	
CBQ	class based queueing
CBRQ	class based random queueing
ccdf	complementary *cdf*
cdf	cumulative density function
CDM	code division multiplexing
CDN	content distribution network
CGE	carrier grade Ethernet
CoS	class of service
Cov	covariance

c_X	coefficient of variation
D	
Diff-Serv	Differentiated services
DPS	discriminatory PS
DVD	digital versatile disk
E	
EO	electrical to optical conversion
EPS	egalitarian PS
ES	egalitarian sharing
F	
FCFS	first come first served
FDM	frequency division multiplexing
FEC	forward error correction
FEC	forwarding equivalence class
FIFO	first in first out
FR	frame relay
FTM	flow transfer mode
G	
Geo	geometric distribution
GMPLS	generalised MPLS
GPD	generalised Pareto distribution
GPS	generalised PS
H	
HD	high definition
HDTV	high definition television
HQS	hierarchic queueing stack
HTB	hierarchical token bucket
I	
IETF	internet engineering task force
ID	identifier
iMAX	image maximum (motion picture format)
IP	internet protocol
IPP	interrupted Poisson process
ISP	internet service provider
IT	information technology
ITU	international telecommunication union
i.i.d.	independent and identically distributed
J	
JET	just enough time
K	
L	
LAN	local area network
LB	leaky bucket
LCFS	last come first served
LDP	label distribution protocol
LER	label edge router
LIFO	last in first out
LSP	label switched path
LTE	long term evolution
M	
MAM	matrix analytic method
MAP	Markov arrival process
MGM	matrix geometric method
MILP	mixed integer linear programming
MMAP	decomposed MAP
MMPP	Markov modulated Poisson process
MOS	mean opinion score
MPLS	multi protocol label switching
MRP	Markov renewal process
m-code	Matlab programming language
N	
NGN	next generation network
NOC	network operation centre
NP	non-deterministic polynomial-time
NP-hard	NP computation complexity
np-LCFS	non preemptive LCFS
O	
OBS	optical burst switching
OCS	optical circuit switching
OE	optical to electric conversion
OEO	electrical processing of optical signals
ON	optical network
OPS	optical packet switching
OSI	open systems interconnection
OSPF	open shortest path first
OTDM	optical TDM
OTH	optical digital hierarchy
OTN	optical transport network
P	
PASTA	Poisson arrivals see time averages
PATON	polymorphous agile transparent ON
P_b	probability of blocking
PCE	path computation element
pdf	probability density function
PESQ	perceptual evaluation of speech quality
PHB	per hop behaviour
pmf	probability mass function
Prio	prioritised
PS	processor sharing

Q
QBD quasi-birth-and-death
QoE quality of experience
QoS quality of service
QoT quality of transmission

R
RAND random queueing
RED random early detection
RF radio frequency
RFC request for comments
RR round robin
RSVP resource reservation protocol
RSVP-TE RSVP for traffic engineering
RTT round trip time
RX receiver, reception rate

S
SDH synchronous digital hierarchy
SDI serial digital (video) interface
SDN software defined networking
SDTV standard definition television
SLA service level agreement
Sonet synchronous optical networking
SP strict priority
SPQ strict priority queueing
SQ shared queue

T
TB token bucket
TCP transport control protocol
TDM time division multiplexing
TE traffic engineering
TX transmitter, transmission rate

U
UDP user datagram protocol
UMTS universal mobile telecom. system

V
Var variance
VC virtual connection
VoIP voice over IP
VP virtual path
VPLS virtual private LAN service

W
WDM wavelength division multiplexing
WFQ weighted fair queueing
WLAN wireless LAN
WRED weighted random early detection
WRQ weighted random queueing
WRQ+ a better WRQ realisation
WRR weighted round robin

X
Y
Z

Bibliography

[1] *ITU-T Recommendation X.200 (1994 E): Information Technology – Open Systems Interconnection – basic reference model: The basic model*, ITU-T, 07/1994, also published as ISO/IEC International Standard 7498-1. URL: http://www.itu.int/rec/T-REC-X.200-199407-I

[2] *ITU-T Recommendation Y.2001: Next Generation Networks - Frameworks and functional architecture models: General overview of NGN*, ITU-T study group 13, 12/2004. URL: http://www.itu.int/rec/T-REC-Y.2001-200412-I

[3] P. Raj, A. Padmapriya, S. Gopalan, and S. Charles, "Achieving balanced traffic distribution in MPLS networks," in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 8, july 2010, pp. 351–355. DOI: 10.1109/ICCSIT.2010.5564831

[4] *IETF RFC-3630: Traffic Engineering (TE) Extensions to OSPF Version 2*, 2003. URL: http://tools.ietf.org/html/rfc3630

[5] *IETF RFC-3031: Multiprotocol Label Switching Architecture*, 2001. URL: http://tools.ietf.org/html/rfc3031

[6] D. O. Awduche and B. Jabbari, "Internet traffic engineering using multi-protocol label switching (MPLS)," *Comput. Netw.*, vol. 40, no. 1, pp. 111–129, Sep. 2002. DOI: 10.1016/S1389-1286(02)00269-4

[7] *IETF RFC-3209: RSVP-TE: Extensions to RSVP for LSP Tunnels*, 2001. URL: http://tools.ietf.org/html/rfc3209

[8] *IETF RFC-2475: An Architecture for Differentiated Services*, 1998. URL: http://tools.ietf.org/html/rfc2475

[9] *IETF RFC-3036: LDP Specification*, 2001. URL: http://tools.ietf.org/html/rfc3036

[10] H. R. van As, "Time for a change in electronic and photonic switching," in *Transparent Optical Networks, 2008. ICTON 2008. 10th Anniversary International Conference on*, vol. 1, June 2008, pp. 140–143. DOI: 10.1109/ICTON.2008.4598391

[11] H. R. van As, "Flow transfer mode (FTM) as universal switching method in electronic and photonic networks," *Elektrotechnik und Informationstechnik (e&i)*, vol. 126, no. 7/8, pp. 301–304, 2009, springer, ISSN: 0932-383X/1613-7620. DOI: 10.1007/s00502-009-0656-y

[12] H. R. van As, "Driving optical network innovation by extensively using transparent domains," in *Transparent Optical Networks (ICTON), 2010 12th International Conference on*, June 2010, pp. 1–4. DOI: 10.1109/ICTON.2010.5549173

[13] G. Franzl, M. F. Hayat, T. Holynski, and A. V. Manolova, "Burst-switched optical networks supporting legacy and future service types," *Journal of Computer Networks and Communications*, vol. 2011, no. article ID 310517, p. 17, 2011. DOI: 10.1155/2011/310517

[14] L. Kleinrock, *Queueing Systems – Volume 1: Theory*. Wiley-Interscience, 1975. ISBN: 978-0471491101

[15] J. Abate and W. Whitt, "The fourier-series method for inverting transforms of probability distributions," *Queueing Systems*, vol. 10, pp. 5–88, 1992. DOI: 10.1007/bf01158520

[16] J. Abate, G. L. Chooudhury, and W. Whitt, "An introduction to numerical transform inversion and its application to probability models," in *Computational probability*, 2nd ed., ser. International Series in Operations Research & Management Science, W. Grassmann, Ed. Boston, USA: Kluwer, 1999, pp. 257–323. ISBN: 978-0792386179

[17] J. Abate and W. Whitt, "A unified framework for numerically inverting Laplace transforms," *INFORMS J. on Computing*, vol. 18, pp. 408–421, January 2006. DOI: 10.1287/ijoc.1050.0137

[18] J. Abate and W. Whitt, "An operational calculus for probability distributions via Laplace transform," *Advances in Applied Probability*, vol. 28, no. 1, pp. 75–113, 1996. URL: http://www.jstor.org/stable/1427914

[19] A. G. Rossberg, "Laplace transforms of probability distributions are easy on logarithmic scales," *J. App. Prob.*, vol. 45, no. 2, pp. 531–541, 2008. URL: http://www.jstor.org/stable/27595962

[20] A. M. Cohen, *Numerical Methods for Laplace Transform Inversion*, ser. C. Brezinski, editor, Numerical Methods and Algorithms Series. Springer, 2007, vol. 5. ISBN: 978-0-387-28261-9. DOI: 10.1007/978-0-387-68855-8

[21] P. G. Massouros and G. M. Genin, "Algebraic inversion of Laplace transform," *Computers & Mathematics with Applications*, vol. 50, pp. 179–185, 2005. DOI: 10.1016/j.camwa.2004.11.017

[22] R. L. Cruz, "A calculus for network delay: Parts I and II," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 114–141, 1991, DOI: 10.1109/18.61109. DOI: 10.1109/18.61110

[23] J.-Y. Le Boudec and A. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, ser. LNCS. Springer, 2001, vol. 2050. ISBN: 978-3540421849

[24] Y. Jiang and Y. Liu, *Stochastic Network Calculus*. Springer, 2008. ISBN: 978-1-84800-126-8. DOI: 10.1007/978-1-84800-127-5

[25] R. Vallejos, A. Zapata, and M. Aravena, "Fast blocking probability evaluation of end-to-end optical burst switching networks with non-uniform ON-OFF input traffic model," *Photonic Network Communications*, vol. 13, no. 2, pp. 217–226, 2007. DOI: 10.1007/s11107-006-0037-y

[26] G. Bloch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains: modeling and performance evaluation with computer science application*, 2nd ed. Wiley-Interscience, 2006. ISBN: 978-0-471-56525-3

[27] M. F. Neuts, "Matrix-analytic methods in queuing theory," *European Journal of Operational Research*, vol. 15, no. 1, pp. 2–12, 1984. DOI: 10.1016/0377-2217(84)90034-1

[28] A. O. Allen, *Probability, Statistics, and Queueing Theory*, 2nd ed., ser. Computer Science and Scientific Computing. Academic Press, 1990. ISBN: 978-0120510511

[29] J. W. Eaton, D. Bateman, and S. Hauberg, *GNU Octave Manual*, 3rd ed., print and on-line, Free Software Foundation, Inc., 2008. ISBN: 978-0-9546120-6-1. URL: https://www.gnu.org/software/octave/doc/interpreter/index.html

[30] J. Y. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*, ser. Engineering and Computer Science. Springer, 1990. ISBN: 978-0-7923-9061-9

[31] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*, 4th ed., ser. Electrical and Computer Engineering. McGraw-Hill Higher Education, 2002. ISBN: 978-0-07-366011-0, 0-07-122661-3

[32] M. Kendall and A. Stuart, *The advanced theory of statistics, Bd. 1*, ser. The Advanced Theory of Statistics. Griffin, 1967. URL: http://books.google.at/books?id=QhnvAAAAMAAJ

[33] A. Stuart, K. Ord, and S. Arnold, *Kendall's Advanced Theory of Statistics, Bd. 2: Classical Inference and the Linear Model*, ser. Kendall's library of statistics. John Wiley & Sons, 2009. ISBN: 9780340662304. URL: http://books.google.at/books?id=3h1W1JCBvN0C

[34] W. J. Stewart, *Probability, Markov Chains, Queues, and Simulation – The Mathematical Basis of Performance Modeling*. Woodstock, Oxfordshire, United Kingdom: Princeton University Press, 2009. ISBN: 978-0-691-14062-9

[35] D. Cox and D. Hinkley, *Theoretical Statistics*. Taylor & Francis, 1979. ISBN: 9780412161605. URL: http://books.google.at/books?id=ppoujo-BInsC

[36] K. Bengi, *Optical Packet-Switched WDM Local/Metro Networks and their Access Protocols*, ser. PhD Thesis. Austria: Vienna University of Technology, 2001. ISBN: 1-4020-7042-X, 978-1402070426

[37] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*, ser. Monographs on Statistics and Applied Probability. Chapman & Hall / CRC Press LLC, 1994. ISBN: 9780412042317. URL: http://books.google.at/books?id=gLlpIUxRntoC

[38] V. G. Cerf and R. E. Khan, "A protocol for packet network intercommunication," *IEEE Transactions on communications*, vol. 22, pp. 637–648, 1974. DOI: 10.1145/1064413.1064423

[39] "Huawei touts prototype flexible grid WDM," on-line, 2012, last accessed 2014/12. URL: http://www.lightwaveonline.com/articles/2012/05/huawei-touts-prototype-flexible-grid-wdm.html

[40] C. Qiao, W. Wei, and X. Liu, "Extending generalized multiprotocol label switching (GMPLS) for polymorphous, agile, and transparent optical networks (PATON)," *IEEE Communication. Magazine*, vol. 44, no. 12, pp. 104–114, 2006. DOI: 10.1109/MCOM.2006.273106

[41] M. Yoo and C. Qiao, "Just-enough time (JET): A high-speed protocol for bursty traffic in optical networks," in *Conf. Tech. Global Info. Infrastructure*. Montreal, Canada: IEEE/LEOS, 1997, pp. 26–27. DOI: 10.1109/LEOSST.1997.619129

[42] P. Pavon-Marino and F. Neri, "On the myths of optical burst switching," *Communications, IEEE Transactions on*, vol. 59, no. 9, pp. 2574–2584, September 2011. DOI: 10.1109/TCOMM.2011.063011.100192

[43] M. F. Neuts, "A versatile markovian point process," *Journal of Applied Probability*, vol. 16, no. 4, pp. 764–779, 1979, alternative URL: http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA056051. URL: http://www.jstor.org/stable/3213143

[44] M. A. Stephens, "EDF statistics for goodness of fit and some comparisons," *Journal of the American Statistical Association*, vol. 69, no. 347, pp. 730–737, 1974. DOI: 10.1080/01621459.1974.10480196

[45] A. Kolmogorov, "Sulla determinazione empirica di una legge di distribuzione," *Giornale dell'Istituto Italiano degli Attuari*, vol. 4, no. 1, pp. 83–91, 1933. DOI: 10.1007/BF01452829

[46] T. W. Anderson, "On the distribution of the two-sample Cramér-von Mises criterion," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1148–1159, 1962. DOI: 10.1214/aoms/1177704477

[47] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *Philosophical Magazine Series 5*, vol. 50, no. 302, pp. 157–175, 1900. DOI: 10.1080/14786440009463897

[48] T. W. Anderson and D. A. Darling, "Asymptotic theory of certain "goodness of fit" criteria based on stochastic processes," *The Annals of Mathematical Statistics*, vol. 23, no. 2, pp. 193–212, 1952. DOI: 10.1214/aoms/1177729437

[49] E. S. Pearson and H. Hartley, *Biometrika tables for statisticians. edited by E.S. Pearson and H.O. Hartley. - Volume 2*. Cambridge, Eng. : Cambridge University Press, 1972, published for the Biometrika Trustees from tables originally published in Biometrika. ISBN: 0521069378

[50] F. Scholz and M. Stephens, "K-sample Anderson–Darling tests," *Journal of the American Statistical Association*, vol. 82, no. 399, pp. 918–924, 1987. URL: http://www.jstor.org/stable/2288805

[51] S. Shapiro, *How to test normality and other distributional assumptions*, ser. ASQC basic references in quality control. American Society for Quality Control Statistics Division, 1990. ISBN: 9780873891042. URL: http://asq.org/quality-press/display-item/index.html?item=E3503

[52] W. Park and Y. Kim, "Goodness-of-fit tests for the power-law process," *Reliability, IEEE Transactions on*, vol. 41, no. 1, pp. 107–111, mar 1992. DOI: 10.1109/24.126680

[53] I. Porter, J.E., J. Coleman, and A. Moore, "Modified KS, AD, and C-vM tests for the Pareto distribution with unknown location and scale parameters," *Reliability, IEEE Transactions on*, vol. 41, no. 1, pp. 112–117, mar 1992. DOI: 10.1109/24.126681

[54] A. Feldmann and W. Whitt, "Fitting mixtures of exponentials to long-tail distributions to analyze network performance models," in *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, apr 1997, pp. 1096–1104 vol.3. DOI: 10.1109/INFCOM.1997.631130

[55] R. Marie, "Calculating equilibrium probabilities for $\lambda(n)/C_k/1/N$ queues," *SIGMETRICS Perform. Eval. Rev.*, vol. 9, no. 2, pp. 117–125, May 1980. DOI: 10.1145/1009375.806155

[56] M. H. Phung, D. Shan, K. C. Chua, and G. Mohan, "Performance analysis of a bufferless OBS node considering the streamline effect," *IEEE Communications Letters*, vol. 10, no. 4, pp. 293–295, 2006. DOI: 10.1109/LCOMM.2006.04006

[57] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of Queueing Theory*, ser. Wiley Series in Probability and Statistics. John Wiley & Sons, 2008, no. 4th edition. ISBN: 047179127X, 978-0471791270. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118211642.html

[58] W. Whitt, "Approximating a point process by a renewal process, I: Two basic methods," *Operations Research*, vol. 30, no. 1, pp. 125–147, 1982. DOI: 10.1287/opre.30.1.125

[59] P. Kuehn, "Approximate analysis of general queuing networks by decomposition," *Communications, IEEE Transactions on*, vol. 27, no. 1, pp. 113–126, 1979. DOI: 10.1109/TCOM.1979.1094270

[60] A. Borny, "Performance analysis of vacation queueing systems and their applications in wireless networks," Ph.D. dissertation, Vienna University of Technology, e389, 2013.

[61] H. Hassan, J.-M. Garcia, and O. Brun, "Generic modeling of multimedia traffic sources," in *Proceedings HET-NET'05, Ilkley, GB*, 2005.

[62] H. C. Tijms, *Stochastic Modelling and Analysis: a computational approach*, ser. Wiley series in probability and mathematical statistics. John Wiley & Suns Ltd., 1986. ISBN: 0-471-909114

[63] E. Gelenbe and G. Pujolle, *Introduction to Queueing Networks*, 2nd ed. Chichester, UK: John Wiley & Sons Ltd, 1998. ISBN: 0-471-96294-5

[64] O. C. Ibe, *Fundamentals of Stochastic Networks*, ser. Book Series. Hoboken, New Jersey: John Wiley & Sons Ltd, 2011. ISBN: 978-1-118-06567-9

[65] U. N. Bhat, *An Introduction to Queueing Theory*, ser. Modeling and Analysis in Applications. Boston, Basel, Berlin: Birkhäuser, 2008. ISBN: 978-0-8176-4724-7. DOI: 10.1007/978-0-8176-4725-4

[66] A. Jensen, "An elucidation of A.K. Erlang's statistical works through the theory of stochastic processes," in *The Life and Works af A.K. Erlang*, E. Brockmeyer, H. Halstroem, and A. Jensen, Eds. Koebenhavn, Danmark: Erlangbogen, 1948, pp. 23–100.

[67] A. J. E. M. Janssen and J. S. H. Van Leeuwaarden, "Back to the roots of the *M/D/s* queue and the works of Erlang, Crommelin and Pollaczek," *Statistica Neerlandica*, vol. 62, no. 3, pp. 299–313, 2008. DOI: 10.1111/j.1467-9574.2008.00395.x

[68] C. D. Crommelin, "Delay probability formulae," *Post Office Electrical Engineers Journal*, vol. 26, pp. 266–274, 1934.

[69] G. J. Franx, "A simple solution for the *M/D/c* waiting time distribution," *Operations Research Letters*, vol. 29, pp. 221–229, 2001. DOI: 10.1016/S0167-6377(01)00108-0

[70] C. D. Crommelin, "Delay probability formulae when the holding times are constant," *Post Office Electrical Engineers Journal*, vol. 25, pp. 41–50, 1932.

[71] J. W. Cohen, *The single server queue*, 2nd ed. North Holland, 1992. ISBN: 978-0444894829

[72] J. F. C. Kingman, "The single server queue in heavy traffic," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 57, no. 4, pp. 902–904, okt 1961. DOI: 10.1017/S0305004100036094

[73] M. Neuts, *Matrix-geometric Solutions in Stochastic Models: An Algorithmic Approach*, ser. Dover books on advanced mathematics. Dover Publications, 1981. ISBN: 9780486683423. URL: http://books.google.at/books?id=WPol7RVptz0C

[74] G. Latouche and V. Ramaswami, "A logarithmic reduction algorithm for quasi-birth-death processes," *Journal of Applied Probability*, vol. 30, no. 3, pp. 650–674, 1993. URL: http://www.jstor.org/stable/3214773

[75] M. Sommereder, "Advanced markov chain techniques in queueing networks," Ph.D. dissertation, Vienna University of Technology, e388, 2009.

[76] M. Sommereder, *Modelling of Queueing Systems with Markov Chains*. Books on Demand, 2011. ISBN: 9783842348431

[77] M. W. Kutta, “Beitrag zur näherungweisen Integration totaler Differentialgleichungen,” Ph.D. dissertation, Ludwig-Maximilians-Universität München, 1901. URL: http://www.genealogy.math.ndsu.nodak.edu/id.php?id=62105

[78] A. C. Hindmarsh, “ODEPACK, a systematized collection of ODE solvers,” *IMACS Transactions on Scientific Computation*, vol. 1, pp. 55–64, 1983, on-line: http://people.sc.fsu.edu/~jburkardt/f77_src/odepack/odepack.html. URL: http://computation.llnl.gov/casc/odepack/

[79] T. O. Engset, *Die Wahrscheinlichkeitsrechnung zur Bestimmung der Waehleranzahl in automatischen Fernsprechaemtern*. Tapir Academic Press, 1918, vol. 39, no. 31, pp. 304–306, translation 1992: "The probability calculation to determine the number of switches in automatic telephone exchanges." by E. Jensen, in Telektronikk 88, 90-93, (Oslo); reprinted in Myskja & Espvik, *Tore Olaus Engset 1865-1943: The Man Behind the Formula*, 2002, pp. 175–183. ISBN: 978-82-519-1828-2

[80] J. Zhang, E. W. M. Wong, and M. Zukerman, “Modeling an OBS node under critical load and high utilization conditions,” *Communications Letters, IEEE*, vol. 16, no. 4, pp. 544–546, april 2012. DOI: 10.1109/LCOMM.2012.030512.112537

[81] M. Zukerman, “Introduction to queueing theory and stochastic teletraffic models,” on-line, Melbourne, Australia, 2000-2012, url: http://www.ee.unimelb.edu.au/staff/mzu/classnotes.pdf.

[82] R. Parkinson, *Traffic Engineering Techniques in Telecommunications*, Infotel Systems Corporation, April 2002, on line, last accessed 02/2014. URL: http://www.tarrani.net/mike/docs/TrafficEngineering.pdf

[83] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems*. New York, USA: Cambridge University Press, 2013. ISBN: 978-1-107-02750-3

[84] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, “Open, closed, and mixed networks of queues with different classes of customers,” *Journal of the ACM*, vol. 22, no. 2, pp. 248–260, Apr. 1975. DOI: 10.1145/321879.321887

[85] L. Kleinrock, *Queueing systems – Volume 2: Computer applications*, ser. Wiley-Interscience Publication. Wiley, 1976. ISBN: 978-0471491118. URL: http://books.google.at/books?id=Pw9DAQAAIAAJ

[86] S. C. Borst, O. J. Boxma, J. A. Morrison, and R. NúñEz Queija, “The equivalence between processor sharing and service in random order,” *Oper. Res. Lett.*, vol. 31, no. 4, pp. 254–262, Jul. 2003. DOI: 10.1016/S0167-6377(03)00006-3

[87] J. W. Cohen, “The multiple phase service network with generalized processor sharing,” *Acta Informatica*, vol. 12, no. 3, pp. 245–284, 1979. DOI: 10.1007/BF00264581

[88] A. Parekh and R. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: the single-node case,” *Networking, IEEE/ACM Transactions on*, vol. 1, no. 3, pp. 344–357, Jun. 1993. DOI: 10.1109/90.234856

[89] A. Parekh and R. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: the multiple node case,” *Networking, IEEE/ACM Transactions on*, vol. 2, no. 2, pp. 137–150, Apr. 1994. DOI: 10.1109/90.298432

[90] S. Aalto, U. Ayesta, S. Borst, V. Misra, and R. Núñez Queija, “Beyond processor sharing,” *SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 4, pp. 36–43, Mar. 2007. DOI: 10.1145/1243401.1243409

[91] E. Altman, K. Avrachenkov, and U. Ayesta, "A survey on discriminatory processor sharing," *Queueing Systems*, vol. 53, no. 1-2, pp. 53–63, 2006. DOI: 10.1007/s11134-006-7586-8

[92] L. Kleinrock, "Time-shared systems: A theoretical treatment," *Journal of the Association for Computing Machinery (ACM)*, vol. 14, no. 2, pp. 242–261, 1967. DOI: 10.1145/321386.321388

[93] G. Fayolle, I. Mitrani, and R. Iasnogorodski, "Sharing a processor among many job classes," *Journal of the Association for Computing Machinery (ACM)*, vol. 27, no. 3, pp. 519–532, Jul. 1980. DOI: 10.1145/322203.322212

[94] S. Borst, O. Boxma, and N. Hegde, "Sojourn times in finite-capacity processor-sharing queues," in *Next Generation Internet Networks, 2005*, April 2005, pp. 53–60. DOI: 10.1109/NGI.2005.1431647

[95] N. Akar, "Moments of conditional sojourn times in finite capacity *M/M/1/N*-PS processor sharing queues," *Communications Letters, IEEE*, vol. 16, no. 4, pp. 533–535, April 2012. DOI: 10.1109/LCOMM.2012.020212.112310

[96] A. B. Bondi and J. P. Buzen, "The response times of priority classes under preemptive resume in *M/G/M* queues," *SIGMETRICS Perform. Eval. Rev.*, vol. 12, no. 3, pp. 195–201, Jan. 1984. DOI: 10.1145/1031382.809328

[97] M. Harchol-Balter, T. Osogami, A. Scheller-Wolf, and A. Wierman, "Multi-server queueing systems with multiple priority classes," *Queueing Syst. Theory Appl.*, vol. 51, no. 3-4, pp. 331–360, Dec. 2005. DOI: 10.1007/s11134-005-2898-7

[98] H. R. Gail, S. L. Hantler, and B. A. Taylor, "Analysis of a non-preemptive priority multiserver queue," *Advances in Applied Probability*, vol. 20, no. 4, pp. 852–879, 1988. URL: http://www.jstor.org/stable/1427364

[99] H. R. Gail, S. L. Hantler, and B. A. Taylor, "On a preemptive markovian queue with multiple servers and two priority classes," *Math. Oper. Res.*, vol. 17, no. 2, pp. 365–391, May 1992. DOI: 10.1287/moor.17.2.365

[100] E. P. Kao and S. D. Wilson, "Analysis of nonpreemptive priority queues with multiple servers and two priority classes," *European Journal of Operational Research*, vol. 118, no. 1, pp. 181–193, 1999. DOI: 10.1016/S0377-2217(98)00280-X

[101] E. Peköz, "Optimal policies for multi-server non-preemptive priority queues," *Queueing Systems*, vol. 42, no. 1, pp. 91–101, 2002. DOI: 10.1023/A:1019946812169

[102] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 4, pp. 1–12, Aug. 1989. DOI: 10.1145/75247.75248

[103] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," *SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 4, pp. 19–29, Aug. 1990. DOI: 10.1145/99517.99525

[104] L. Wang, G. Min, D. Kouvatsos, and X. Jin, "An analytical model for the hybrid PQ-WFQ scheduling scheme for WiMAX networks," in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*, May 2009, pp. 492–498. DOI: 10.1109/WIRELESSVITAE.2009.5172494

[105] A. Kumar, D. Manjunath, and J. Kuri, *Communication Networking: An Analytical Approach*, ser. The Morgan Kaufmann series in networking. Elsevier/Morgan Kaufmann, 2004. ISBN: 978-0124287518. URL: http://books.google.at/books?id=z2IzMZn4I4EC

[106] J. Zhang, T. Hamalainen, and J. Joutsensalo, "Stochastic analysis of upper delay bound of GPS-based packetized fair queueing algorithms," in *Networks, 2004. (ICON 2004). Proceedings. 12th IEEE International Conference on*, Nov 2004, pp. 35–40 vol.1. DOI: 10.1109/ICON.2004.1409082

[107] D. Stiliadis and A. Varma, "Rate-proportional servers: a design methodology for fair queueing algorithms," *Networking, IEEE/ACM Transactions on*, vol. 6, no. 2, pp. 164–174, Apr. 1998. DOI: 10.1109/90.664265

[108] T. Balogh and M. Medvecký, "Average bandwidth allocation model of WFQ," *Modelling and Simulation in Engineering*, vol. 2012, no. 301012, Nov. 2012. DOI: 10.1155/2012/301012

[109] C. Semeria, "Supporting differentiated service classes: Queue scheduling disciplines," *Juniper Networks, Inc. white paper*, last accessed 02/2015, 2001. URL: http://users.jyu.fi/~timoh/kurssit/verkot/scheduling.pdf

[110] M. Channegowda, R. Nejabati, and D. Simeonidou, "Software-defined optical networks technology and infrastructure: Enabling software-defined optical network operations [invited]," *J. Opt. Commun. Netw.*, vol. 5, no. 10, pp. A274–A282, Oct 2013. DOI: 10.1364/JOCN.5.00A274

[111] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, Jul. 1997. DOI: 10.1145/263932.264023

[112] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *ACM SIGCOMM Computer Communication Review*, vol. 33, pp. 83–91, 2003. DOI: 10.1145/956981.956989

[113] E. S. Hashem, *Analysis of random drop for gateway congestion control*, ser. master thesis, Report LCS TR-465, Laboratory for Computer Science, MIT. Cambridge, MA, USA: DIANE Publishing, 1989. ISBN: 9781428983182. URL: http://books.google.co.uk/books?id=4qukjdkuBGwC

[114] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking (TON)*, vol. 1, no. 4, pp. 397–413, Aug. 1993. DOI: 10.1109/90.251892

[115] V. Jacobson, K. Nichols, and K. Poduri, "RED in a different light," unpublished draft paper, 1999. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.9406

[116] G. Feng, A. Agarwal, A. Jayaraman, and C. K. Siew, "Modified RED gateways under bursty traffic," *Communications Letters, IEEE*, vol. 8, no. 5, pp. 323–325, May 2004. DOI: 10.1109/LCOMM.2004.827427

[117] P. Tinnakornsrisuphap and R. J. La, "Asymptotic behavior of heterogeneous TCP flows and RED gateway," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 108–120, 2006. DOI: 10.1109/TNET.2005.863453

[118] S. Linck, E. Dedu, and F. Spies, "Distance-dependent RED policy (DDRED)," in *Networking, 2007. ICN '07. Sixth International Conference on*. Los Alamitos, CA, USA: IEEE Computer Society, April 2007, pp. 51–51. ISBN: 0-7695-2805-8. DOI: 10.1109/ICN.2007.37

[119] A. Jukan and G. Franzl, "Path selection methods with multiple constraints in service-guaranteed WDM networks," *IEEE-ACM Transactions on Networking*, vol. 12, no. 1, pp. 59–72, 2004. DOI: 10.1109/TNET.2003.822657

[120] *IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks — Amendment 9: Stream Reservation Protocol (SRP)*, 09/2010. URL: http://www.ieee802.org/1/pages/802.1at.html

[121] A. Elwalid and D. Mitra, "Effective bandwidth of general markovian traffic sources and admission control of high speed networks," *Networking, IEEE/ACM Transactions on*, vol. 1, no. 3, pp. 329–343, Jun. 1993. DOI: 10.1109/90.234855

[122] J. S. Turner, "New directions in communications (or which way to the information age?)," *IEEE Communications Magazine*, vol. 24, no. 10, pp. 8–15, 1986. DOI: 10.1109/MCOM.1986.1092946

[123] Z.-X. Zhao, S. Panwar, and D. Towsley, "Queueing performance with impatient customers," in *INFOCOM '91. Proceedings. Tenth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking in the 90s., IEEE*, Apr 1991, pp. 400–409 vol.1. DOI: 10.1109/INFCOM.1991.147530

[124] J. Boyer, F. Guillemin, P. Robert, and B. Zwart, "Heavy tailed M/G/1-PS queues with impatience and admission control in packet networks," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, March 2003, pp. 186–195 vol.1. DOI: 10.1109/INFCOM.2003.1208671

[125] K. Wang, N. Li, and Z. Jiang, "Queueing system with impatient customers: A review," in *Service Operations and Logistics and Informatics (SOLI), 2010 IEEE International Conference on*, July 2010, pp. 82–87. DOI: 10.1109/SOLI.2010.5551611

[126] J. Kim, B. Kim, and J. Kang, "Discrete-time multiserver queue with impatient customers," *Electronics Letters*, vol. 49, no. 1, pp. 38–39, January 2013. DOI: 10.1049/el.2012.3520

[127] M. Devera, *Hierachical token bucket theory*, 5.5.2002, last retrieved 22. Oct. 2014. URL: http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm

[128] G. Franzl, "Interdependence of dynamic traffic flows," in *Transparent Optical Networks, 2008. ICTON 2008. 10th Anniversary International Conference on*, vol. 3, June 2008, pp. 186–189. DOI: 10.1109/ICTON.2008.4598686

[129] B. Statovci-Halimi and G. Franzl, "Future internet services: Tiered or neutral," *E&I Elektrotechnik und Informationstechnik*, vol. 128, no. 10, pp. 366–370, 2011. DOI: 10.1007/s00502-011-0047-z

[130] B. Statovci-Halimi and G. Franzl, "QoS differentiation and internet neutrality — a controversial issue within the future internet challenge," *Telecommunication Systems*, vol. 52, no. 3, pp. 1605–1614, 2013. DOI: 10.1007/s11235-011-9517-1

[131] P. J. Burke, "The output of a queuing system," *Operations Research*, vol. 4, no. 6, pp. 699–704, 1956. DOI: 10.1287/opre.4.6.699

[132] E. Reich, "Waiting times when queues are in tandem," *The Annals of Mathematical Statistics*, vol. 28, no. 3, pp. 768–773, 09 1957. DOI: 10.1214/aoms/1177706889

[133] J. R. Jackson, "Networks of waiting lines," *Operations Research*, vol. 5, no. 4, pp. 518–521, 1957. DOI: 10.1287/opre.5.4.518

[134] P. M. Morse, "Stochastic properties of waiting lines," *Journal of the Operations Research Society of America*, vol. 3, no. 3, pp. 255–261, 1955. DOI: 10.1287/opre.3.3.255

[135] N. Bean, D. Green, and P. Taylor, "The output process of an MMPP/M/1 queue," *Journal of Applied Probability*, vol. 35, no. 4, pp. 998–1002, 12 1998. DOI: 10.1239/jap/1032438394

[136] J. Walrand and P. Varaiya, "Sojourn times and the overtaking condition in Jacksonian networks," *Advances in Applied Probability*, vol. 12, no. 4, pp. 1000–1018, 12 1980. DOI: 10.2307/1426753

[137] W. J. Gordon and G. F. Newell, "Closed queuing systems with exponential servers," *Operations Research*, vol. 15, no. 2, pp. 254–265, 1967. DOI: 10.1287/opre.15.2.254

[138] F. P. Kelly, *Reversibility and Stochastic Networks*. New York, NY, USA: Cambridge University Press, 2011. ISBN: 1107401151, 9781107401150

[139] I. Akyildiz and A. Sieber, "Approximate analysis of load dependent general queueing networks," *Software Engineering, IEEE Transactions on*, vol. 14, no. 11, pp. 1537–1545, Nov. 1988. DOI: 10.1109/32.9042

[140] P. J. Hunt and F. P. Kelly, "On critically loaded loss networks," *Advances in Applied Probability*, vol. 21, no. 4, pp. 831–841, 1989. URL: http://www.jstor.org/stable/1427769

[141] S. Söhnlein and A. Heindl, "Analytic computation of end-to-end delay in queueing networks with batch markovian arrival processes and phase-type service times," in *Proc. of 13-th Int. Conf. on Analytical and Stochastic Modelling Techniques and Applications (ASMTA 2006)*, 2006, pp. 28–31. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.743&rep=rep1&type=pdf

[142] A. E. Conway and N. D. Georganas, *Queueing Networks – Exact Computational Algorithms: A Unified Theory Based on Decomposition and Aggregation*. Cambridge, MA, USA: MIT Press, 1989. ISBN: 0-262-03145-0

[143] A. Horváth, G. Horváth, and M. Telek, "A traffic based decomposition of two-class queueing networks with priority service," *Computer Networks*, vol. 53, no. 8, pp. 1235–1248, 2009, performance Modeling of Computer Networks: Special Issue in Memory of Dr. Gunter Bolch. DOI: 10.1016/j.comnet.2009.02.016

[144] *ITU-T Recommendation G.1010: End-user multimedia QoS categories*, ITU-T Study Group 12, 11/2001. URL: http://www.itu.int/rec/T-REC-G.1010/en

[145] *ITU-T Recommendation P.800: Methods for subjective determination of transmission quality*, ITU-T Study Group 12, 08/1996. URL: http://www.itu.int/rec/T-REC-P.800-199608-I/en

[146] *ITU-T Recommendation P.862: Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs*, ITU-T Study Group 12, 02/2001.

[147] L. Xiao, M. Johansson, and S. Boyd, "Simultaneous routing and resource allocation via dual decomposition," *Communications, IEEE Transactions on*, vol. 52, no. 7, pp. 1136–1144, July 2004. DOI: 10.1109/TCOMM.2004.831346

[148] H. Kim and N. Feamster, "Improving network management with software defined networking," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 114–119, February 2013. DOI: 10.1109/MCOM.2013.6461195

[149] N. Foster, A. Guha, M. Reitblatt, A. Story, M. Freedman, N. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison, "Languages for software-defined networks," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 128–134, February 2013. DOI: 10.1109/MCOM.2013.6461197

[150] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? implementation challenges for software-defined networks," *Communications Magazine, IEEE*, vol. 51, no. 7, pp. 36–43, July 2013. DOI: 10.1109/MCOM.2013.6553676

[151] K. Jeong, J. Kim, and Y.-T. Kim, "QoS-aware network operating system for software defined networking with Generalized OpenFlows," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, April 2012, pp. 1167–1174. DOI: 10.1109/NOMS.2012.6212044

[152] R. Bennesby, P. Fonseca, E. Mota, and A. Passito, "An inter-AS routing component for software-defined networks," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, April 2012, pp. 138–145. DOI: 10.1109/NOMS.2012.6211892

[153] *IETF RFC-3945: Generalized Multi-Protocol Label Switching (GMPLS) Architecture*, Internet Engineering Task Force, 2004. URL: http://www.ietf.org/rfc/rfc3945.txt

[154] M. Yoo, C. Qiao, and S. Dixit, "Optical burst switching for service differentiation in the next-generation optical internet," *IEEE Communications Magazine*, vol. 39, no. 2, 2001. DOI: 10.1109/35.900637

[155] H. L. Vu and M. Zukerman, "Blocking probability for priority classes in optical burst switching networks," *IEEE Communications Letter*, vol. 6, pp. 214–216, 2002. DOI: 10.1109/4234.1001668

[156] The omnibus broadband initiative (OBI), *US National Broadband Plan*, on-line, Federal Communications Comission (FCC), mar 2010, last accessed 2015/02. URL: http://download.broadband.gov/plan/national-broadband-plan.pdf

[157] K. Hendling, B. Statovci-Halimi, G. Franzl, and A. Halimi, "A new bandwidth guaranteed routing approach for online calculation of lsps for mpls traffic engineering," in *Management of Multimedia Networks and Services*, ser. Lecture Notes in Computer Science, A. Marshall and N. Agoulmine, Eds. Springer Berlin Heidelberg, 2003, vol. 2839, pp. 220–232. ISBN: 978-3-540-20050-5. DOI: 10.1007/978-3-540-39404-4_17

[158] J. Karvo, "Generalized engset system - properties and an application," in *In Workshop on Modelling, Measuring and Quality of Service – Proc. of the 7th Summer School on Telecommunications*, 1998, pp. 88–97.

[159] J. F. Hayes and T. V. J. Ganesh Babu, *Modeling and Analysis of Telecommunications Networks*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2004. ISBN: 9780471348450, 9780471643500. DOI: 10.1002/0471643505

[160] A. Zalesky, E. Wong, M. Zukerman, and H. Vu, "Engset formula for bufferless OBS/OPS: When is and when isn't lengthening the off-time redundant?" in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, Nov 2009, pp. 1–6. DOI: 10.1109/GLOCOM.2009.5425990

[161] R. J. Boucherie and N. M. van Dijk, Eds., *Queueing Networks: A Fundamental Approach*, ser. International Series in Operations Research & Management Science, vol. 154. Springer, 2011. ISBN: 978-1-4419-6471-7. URL: http://www.springer.com/mathematics/probability/book/978-1-4419-6471-7

[162] J. Suzuki and T. Nakano, Eds., *Bio-Inspired Models of Network, Information, and Computing Systems*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 87. Springer, 2012, revised selected papers from 5th international ICST conference, BIONETICS 2010, Boston, USA, December 1-3, 2010. ISBN: 978-3-642-32614-1 (print) 978-3-642-32615-8 (on-line). URL: http://link.springer.com/book/10.1007/978-3-642-32615-8

[163] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control and Stochastic Networks Perspective*. Cambridge University Press, 2014. ISBN: 9781107036055. URL: http://www.cambridge.org/9781107036055

Note, there exist many more papers and textbooks, and commonly we looked through more then the finally cited publications to learn and understand the presented approaches. Thus, the list of references represents only a selection of the rich literature available, and misses many basic, relevant, important, or very recent works, as for example [158–163].