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

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

institute of telecommunications

Optimized Active Redundancy with Just-in-Time Duplication for Safety Critical Mobile Video Streaming

Diploma Thesis

for obtaining the academic degree

Diplom-Ingenieur

as part of the master's program

Computer Technology

carried out by

Stefan Lendl BSc.

student number: 00928895

Institute of Telecommunications

at TU Wien

Supervision:

Univ.Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby

Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Joachim Fabini

Statement on Academic Integrity

I hereby declare and confirm with my signature that the master's thesis is exclusively the result of my own autonomous work based on my research and literature published, which is seen in the notes and bibliography used. I also declare that no part submitted has been made in an inappropriate way, whether by plagiarizing or infringing on any third person's copyright. Finally, I declare that no part submitted has been used for any other paper in another higher education institution, research institution or educational institution.

Vienna, June 2019

Author's signature

Abstract

In safety-critical applications reliable delivery and timely arrival of data are equally important properties and yet known to be conflicting in real-time multimedia streaming. Whenever real-time systems depend on unreliable communication infrastructure like for instance IP networks, multihoming — the availability of redundant network paths — can provide fault tolerance and significantly increase overall systems reliability.

This thesis examines different transmission strategies for multihomed setups to improve the reliability while at the same time considering latency and link efficiency requirements. Duplicating the entire stream onto multiple paths is simple but adds a significant data overhead. Retransmitting packets merely upon a detected loss, as offered by TCP, lowers data overhead but causes non-deterministic latency characteristics.

Thus, the author of this thesis proposes Deadline Based Preemptive Retransmission (DPR), a transmission scheduling strategy that delays multipath packet duplication to meet the latency requirement just-in-time. This allows for more efficient link utilization by avoiding unnecessary data duplication. DPR is implemented as an extension to Concurrent Multipath Transmission (CMT), which is itself an extension of the Stream Control Transmission Protocol (SCTP).

This thesis assesses the constraints and requirements of real-time transmission as well as the consequences of late packet arrival. This thesis presents new metrics to assess and quantify the performance of transmission strategies to satisfy the requirement of timely delivery of data as well as the penalty of introduced data overhead. The proposed transmission scheduling DPR and other investigated scheduling approaches are validated embedded in a multimedia-streaming pipeline with a modified version of GStreamer and an emulated network environment. The conducted experiments investigate the characteristics, performance and drawbacks of different duplication strategies under various network characteristics. The empirical analysis demonstrates the conspicuous benefits of DPR.

Contents

1 Introduction **2**

1.1 Motivation 2

1.2 Research Questions and Goals 3

1.3 Scope 3

1.4 Recent Work 4

1.5 Structure 4

2 Technical Background **6**

2.1 Terminology 6

2.2 Protocol Layering 8

2.3 Protocol Mechanisms 10

2.3.1 Routing 11

2.3.2 Mobility 11

2.3.3 Connection-Oriented and Connectionless Service 11

2.3.4 Fragmentation 12

2.3.5 Ordered Delivery 12

2.3.6 Reliable Delivery 12

2.3.7 Handling of Data Corruption 13

2.3.8 Flow Control 13

2.3.9 Congestion Control 13

2.4 Path Characteristics 14

2.4.1 Packet Delay 14

2.4.2 Packet Delay Variation 16

2.4.3 Bandwidth Delay Product 16

2.4.4 Packet Loss 16

2.5 Dependability 17

2.6 Real-Time Multimedia Streaming 18

2.6.1 Video Codec 19
2.6.2 Multimedia Streaming Related Protocols 20

3 Requirements Analysis 22
3.1 Environment 22
3.2 Real-Time Streaming Requirements 22
3.3 Connectivity 23
3.4 Fault Model 24
3.5 Redundancy 25
3.5.1 Multihoming 26
3.5.2 Forward Error Correction 28
3.5.3 Interleaving 29
3.5.4 Conclusion 29
3.6 Packet Replication For Multihoming 29
3.6.1 Link Layer Approaches 30
3.6.2 Network Layer Approaches 30
3.6.3 Transport Layer Approaches 31
3.6.4 Application Layer Approaches 31
3.6.5 Conclusion 31
3.7 Transport Layer Protocols 32
3.7.1 User Datagram Protocol 32
3.7.2 Transmission Control Protocol 32
3.7.3 Datagram Congestion Control Protocol 34
3.7.4 Multipath TCP 34
3.7.5 Stream Control Transmission Protocol 35
3.7.6 Comparison and Conclusion 37
3.8 Concurrent Multipath Transmission 38
3.8.1 Buffer Blocking 38
3.9 Conclusion 42

4 Systems Model and Alternatives **43**
4.1 Key Performance Metrics 43
4.1.1 Time To Delivery 43
4.1.2 Delivery Within Deadline Ratio 44
4.1.3 Path Failure Impact 45
4.1.4 Transmission Overhead 45
4.2 Fast Retransmission 46
4.3 Full Packet Duplication 49
4.4 Deadline Based Preemptive Retransmission 49
4.5 Model Verification 53

5 Implementation **54**
5.1 GStreamer Framework 54
5.2 Multimedia Pipeline and Test Setup 55
5.3 GStreamer SCTP Plugin 56
5.4 SCTP Socket 57
5.4.1 usrsctp 57
5.4.2 DPR in usrsctp 58
5.4.3 Modified Congestion Control in usrsctp 59

6 Model Evaluation **60**
6.1 Parameters 61
6.2 Measurements 66
6.2.1 Time To Delivery 67
6.2.2 Delivery Within Deadline Ratio 67
6.2.3 Transmission Overhead 67
6.2.4 Path Failure Impact 68
6.2.5 Test Execution and Evaluation Scripts 69

7 Results **71**

7.1 Delivery Within Deadline Ratio 71

7.1.1 With Modified Congestion Control 73

7.1.2 With Regular Congestion Control 78

7.2 Transmission Overhead 83

7.3 Path Failure Impact 84

8 Conclusion **87**

8.1 Future Work 90

9 References **92**

A Appendix **102**

1 Introduction

1.1 Motivation

This thesis is part of a research project that builds the foundation for later development of autonomous trams for passenger transport in urban areas. The trams will operate without human operators on-board. A 3D-perception via stereo vision algorithm is planned to control the movement of the tram. This algorithm observes the surrounding environment to detect dangerous situations that could lead to collisions in order to protect passengers, pedestrians and other traffic participants. The media captured by the stereo vision camera system will be streamed to the control center.

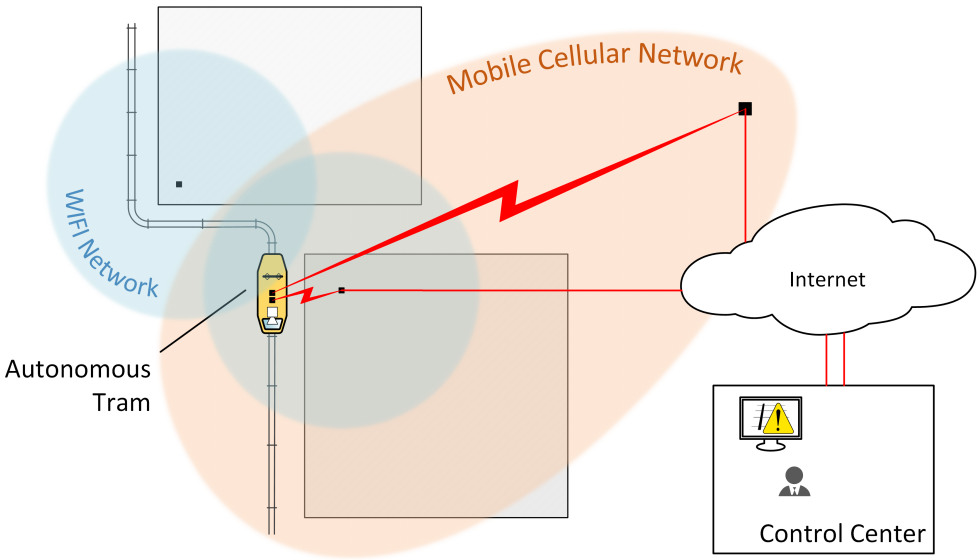

Figure 1: Schematic of the system's environment

To monitor the function and the state of the tram, a stable and continuous data communication to a central station is essential. As presented schematically in Figure 1 the tram is wirelessly connected to the Internet by the use of mobile cellular data technologies like Long Term Evolution (LTE) [1], High Speed Packet Access (HSPA) [2] and — if available in certain areas — Wireless Fidelity (WiFi) [3]. While the tram is driving through the city, the properties of the connection are changing frequently. When the tram for example turns around a corner or enters an underground section, the connection to one of the wireless network cells may suddenly become unstable. The packet loss rate or the packet delay may increase. Resulting connection degradations

may be transient or lead to permanent connection failure. The connection should be sufficiently resilient to withstand all these influences while maintaining a high quality standard for the media transferred. To compensate for these fluctuations in wireless connectivity, multiple wireless links should be combined to increase the redundancy and therefore the reliability of the overall connection to the control center.

1.2 Research Questions and Goals

The goals of this thesis are to investigate the environment and challenges a mobile device, like in this case a tram, is encountering when real-time multimedia is streamed over a heterogeneous network. Network impairments like packet loss and path failure are to be investigated in the context of the multimedia streaming content transferred, which imposes additional requirements on the network. These requirements will be used as a base for decisions regarding technologies and techniques to provide a stable and dependable connection from the mobile device to the central station.

The goal of this thesis is to develop and evaluate a data communication scheme to improve the dependability of the transmitting application by improving the dependability of the underlying data connection.

The following research questions were chosen for the investigation.

- *How can concurrent multipath transmission be used to improve the dependability of a latency constrained real-time multimedia streaming.*
- *How can concurrent transmission on multiple paths be achieved in a single logically related flow and which challenges are encountered in such a case.*
- *How can packet replication improve the fault tolerance of a mobile multi-homed latency-constrained real-time multimedia streaming system.*

1.3 Scope

The techniques to be investigated and improved are limited to data communication-related topics like network protocols, path selection, multihoming, network scheduling, packet loss and error detection as well as error correction.

Multimedia related topics like selection of appropriate multimedia codecs are only investigated briefly. Potential improvements related to the optimization of media coding or decoding are considered out of scope of this thesis.

1.4 Recent Work

Researchers have been investigating various aspects of multipath transport. The authors of [4] analyze in-depth its characteristics and problems, specifically in CMT SCTP and propose various improvements to reduce buffer blocking and increase the throughput.

Researchers in [5] investigated whether multipath transport is suitable for latency sensitive traffic. They conclude that multipath transport improves the overall latency compared to transport on a single path if the paths have comparable path delays. With asymmetrical path delays, CMT SCTP adds significant latency caused by reordering. This stems from the round-robin scheduling in CMT which aims to improve the throughput but does not consider latency. Various path selection and packet scheduling approaches are targeting this issue. For example, Delay-Aware Packet Scheduling (DAPS) [6] and Out-of-order Scheduling In-order Arrival (OSIA) [7] are schedulers for multipath transport that use the Round Trip Time (RTT) of the individual paths and try to send packets in a sequence that guarantees in-order arrival at the receiver. This reduces the receiver buffer blocking caused by reordering.

Various researchers consider optimizations targeted at multimedia streaming. The solution in [8] prioritizes the packets in a video stream according to their decoding priority by using retransmission limits of Partial Reliability (PR) SCTP. Higher priority packets are retransmitted more often. Researchers in [9] propose an extension for SCTP specifically for unreliable multipath transport optimized for video data. It optimizes the sending sequence to allow packet prioritization and allows the receiver to actively abandon packets and reduce receiver buffer blocking by adding a path sequence number to the SCTP header. These solutions result in a throughput improvement and reduces buffering delay for video-on-demand services. In [10] researchers investigate real-time multimedia streaming with low-latency requirements. The approach improves the video quality while meeting the latency constraints by dynamically adapting Forward Error Correction (FEC) parameters of the transmission according to a quality estimation of the decoded video at the receiver.

1.5 Structure

This thesis starts in Section 2 with the description of used terminology and definitions of common concepts in packet switched networking. This includes the protocol layering architecture, certain path characteristics, a definition of dependability and real-time multimedia streaming related concepts like video codecs and relevant protocols. Section 3 examines the environment and requirements of the application of interest in this

thesis and investigates various strategies and suitable networking protocols to increase reliability in a multihomed setup. Section 4 models different packet duplication strategies and defines the key performance metrics used in the experimental verification of the model. Section 5 describes the implementation of the previously defined models in a video streaming pipeline based on GStreamer that uses a modified version of a userspace SCTP implementation. Section 6 explains the measurement setup and describes the calculation of the key performance metrics. Section 7 presents the results of the empirical measurements in form of graphical plots that enable visual support for the comparison of the different duplication strategies in different test environments. This thesis concludes with a short summary of the key findings of this work and gives an outlook of further developments and open research questions in Section 8.

2 Technical Background

Fundamental to the analysis is an exact definition of terms and metrics. This section presents the relevant technical background. It outlines the basic terminology and the reference layer model for protocol development along with some relevant mechanisms offered by these layers, followed by metrics like packet delay or packet loss, that the performance evaluation relies on.

2.1 Terminology

The term *data communications* describes the transfer of data between a sender and a receiver node within a network. The sender transmits the data and the receiver receives it [11].

This thesis is restricted to communications that use principles related to packet switched networking. Such a network consists of many *nodes*, also called *hosts*. The nodes that participate in the communication, the *sender* and the *receiver* are called *endpoints*. From the view of an endpoint the opposite endpoint in a communication is referred to as the *peer*. For the transmission the data is encapsulated into so-called *packets* that propagate from node to node. The physical connection between two nodes is called a *link*. The *path* describes the set of subsequent links and hosts that a packet traverses through the network. The path is usually not predefined and can be different for each packet sent within the same communication session. This means that packets do not necessarily arrive in the sending sequence due to different routes through the network.

Characteristics that influence on the packet transmission are the *link capacity* Γ that indicates the maximum transferred amount of data per time. The actual transferred data volume V within a time Δt is indicated by the *throughput* γ as described in Equation (1). The link with the lowest link capacity on a path is called the *bottleneck* link, which defines the overall link capacity of the path. Equally, a host that limits the overall link capacity of the path, can be a bottleneck.

$$\gamma = \frac{V}{\Delta t} \tag{1}$$

Data communication services can be categorized by the number of participating entities, the number of senders and receivers. A communication between one sender and

one receiver is called *unicast*. A communication between one sender and an unrestricted number of receivers is called *broadcast*. If the reception is restricted to a group of receivers, the communication is called *multicast* [4].

A sequence of related packets between endpoints can be denoted as a flow. In [12] a flow is defined as "*a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination that the source desires to label as a flow. A flow could consist of all packets in a specific transport connection or a media stream. However, a flow is not necessarily 1:1 mapped to a transport connection.*"

To generalize the use of mathematical symbols and abbreviations used in the equations and textual descriptions, Table 1 gives an overview of the symbols used.

Notation	**Description**
Γ	Link Capacity (LC)
γ	Throughput
δ	Delay or latency
δ_e	End-to-end delay
δ_t	Transmission delay
δ_p	Propagation delay
δ_q	Queuing delay
δ_c	Processing delay
δ_R	End-to-end delay including a retransmission
ν	Packet Delay Variation (PDV)
t	Variable for time
D	Deadline
D_R	Retransmission deadline
τ	Point in Time
τ_0	Start of measurement in an experiment
τ_E	End of an experiment
τ_{PF}	Time of a path failure
τ_R	Time a packet is measured at the receiver
τ_S	Time a packet is measured at the sender
RTT	Round Trip Time
$SRTT$	Smoothed Round Trip Time
νRTT	Round Trip Time Variation

Notation	Description
BDP	Bandwidth Delay Product
O	Amount of Missing, Not Yet Acked Bytes
$CWND$	Congestion Window Size
$RWND$	Advertised Receiver Window Size
P	Set Of Packets
p_i	Packet With the Index i
V	Transferred Volume in Bytes
$V(p_i)$, $V(P)$	Size in bytes of a packet or a set of packets
α, β	Smoothing Factors for the SRTT
σ	standard deviation

Table 1: Notations

2.2 Protocol Layering

In order to provide structure and reusability in the design of network protocols, these protocols are organized into different layers. The ISO/OSI reference model [13] defines seven protocol layers with specified functions. The Internet layering architecture [14] defines five layers. This thesis focuses on the Internet layering architecture. Protocols typically span one layer or several layers. Each layer offers a service by performing a certain action within that layer, in interaction with another node, by using the service of the layer below it [15]. Figure 2 shows how the five layers of the Internet layering architecture are organized.

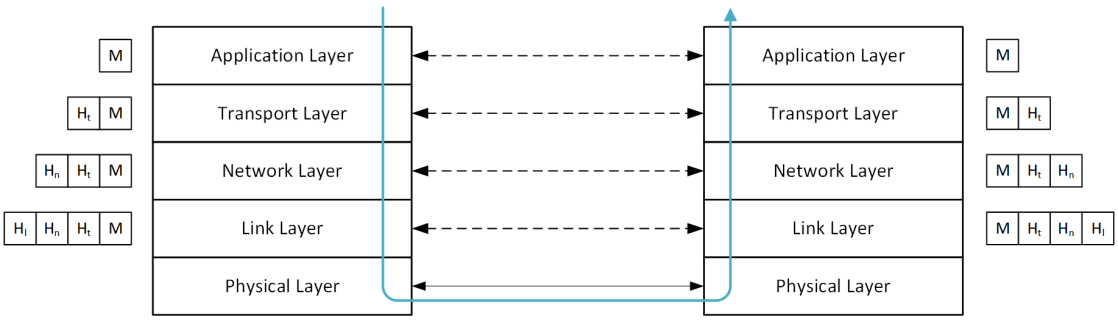

Figure 2: Protocol Layering of the Internet Architecture Model

In order to enable interaction within the layer, the user message is encapsulated by adding layer-specific information in a so-called *header* on each of the layers. At the

receiver, the headers are removed one by one on each layer until the original message is passed on to the application. A specific implementation of these or parts of these layers is called the *network stack*.

Every layer defines its Protocol Data Unit (PDU) which has different names on each layer and varies for the protocols used. This thesis uses the term *packet* to denote the unit of transferred, independently of the protocol. The maximum size of data that can be transferred in one PDU is defined by the Maximum Transfer Unit (MTU) which is defined by the smallest MTU along the path.

Application Layer

Network applications and protocols in the Internet reside in the application layer and use the network stack protocols underneath it. Due to the variety of applications this layer includes many protocols. Prominent examples of application layer protocols are Hypertext Transfer Protocol (HTTP) [16], which provides request and transfer capability Web documents and File Transfer Protocol (FTP) [17] to transfer data files between two endpoints. For multimedia streaming the Real-Time Transport Protocol (RTP) is one of the most important protocols, as it provides important information necessary for playback of the multimedia stream. Section 2.6.2 describes RTP in more depth.

Transport Layer

Transport layer protocols transfer application layer messages between processes on the endpoint systems [15]. To distinguish which process a connection belongs to, transport layer protocols typically define *port* numbers. Typical functions of transport layer protocols are connection establishment, delivery in sequence, congestion control to prevent congestion in network and provide fairness between flows and flow control to prevent the peer from overloading. These and others are described more in-depth in Section 2.3. The most widely used transport layer protocols are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). These and other protocols that are of particular interest the context of this thesis are discussed in more depth in Section 3.7.

Network Layer

Network layer protocols are responsible for transferring packets from one host to another [15].

The most widely used network layer protocol is the Internet Protocol (IP), which exists in version 4 (IPv4) [18] and version 6 (IPv6)[19]. The IP provides addressing for each host, which enables the packet to find its path through the network based on the destination host's address. This so-called destination-based routing is performed by routing protocols in network layer.

Link Layer

Link layer protocols are responsible for transferring data from one node to the next along the path between the endpoints. In every router on the way the link layer header is changed according to the result of the path decision of the network layer protocol.

Link layer protocols are typically responsible to gain access to the underlying medium and checking if a collision between two transmitters occurred. Some protocols provide reliable delivery until the next node. Typical link layer protocols are Ethernet and WiFi [20] protocols to provide wireless local area networking.

Physical Layer

The function of the physical layer is to actually transfer bits on the physical medium. A medium can be a fiber optical cable, a copper wire or the air when transmitting wirelessly. Protocols on this layer exist for each medium that are to be utilized.

2.3 Protocol Mechanisms

This section presents some protocol specific mechanism that are relevant for the discussion.

2.3.1 Routing

In [15] routing is defined as follows: "*Routing refers to the network-wide process that determines the end-to-end paths that packets take from source [sender] to destination [receiver].*"

2.3.2 Mobility

Clients that move from one location to another, are called *mobile*. Typically, IP addresses are associated with a location, not with a client. A mobile client receives a different IP address on every location it connects to. The IP address might change with connection to a different WiFi or LTE cell. Typical connections like with TCP terminate, if an endpoint address changes. With the increasing number of mobile clients, like smartphones, the demand for a new addressing mechanism increases.

Mobility mechanisms allow a client that is connected to one network to move to another network and still be accessible through the same addressing. There are various types of mobility. Typically, this is done by promoting the new address to the peer; tunneling the data from a reference router to the client or hiding the location of the endpoint by providing a different addressing scheme in a form of a client specific identifier. IP offers mobility functionality at transport layer. Locator/Identifier Split Protocol (LISP) as described in Section 3.6.2 offers a client specific, location independent addressing by adding a shim layer between IP and a transport layer protocol on top of it.

2.3.3 Connection-Oriented and Connectionless Service

A *connection-oriented* service establishes a connection, which requires a confirmation from the peer, before sending packets. During connection establishment the endpoints negotiate certain parameters like protocol capabilities, sequence numbers or window sizes. The connection termination usually has a defined procedure as well. An established connection is also called a *session*. A *connectionless* service does not establish a connection first but a sender merely sends out data to its destination. Connectionless protocols usually rely on other protocols or higher layer logic to set up the relation between sender and receiver. Connection or session setup is typically a service of application or transport layer.

2.3.4 Fragmentation

If the application or a protocol intends to send a message that exceeds the MTU of an underlying layer, the message has to be split into small enough segments before passing it on to the next layer. This process is called *fragmentation*. Reassembly of these fragments happens on the same layer at the receiver. Fragmentation is typically done on one or several of application, transport or network layer.

2.3.5 Ordered Delivery

Packets are handed over to a protocol in a certain order. A service that provides ordered delivery ensures that the packets are passed to the upper layer protocols at the receiver in the same order as they were sent by the sender. To reconstruct the original order, the sender typically marks the packets with a sequence number before sending. For reordering the protocol at the receiver typically has to store packets that have arrived *out-of-order* until the missing ones arrive. Protocols with unordered delivery pass the packets in the order of arrival on to the upper layer protocol at the receiver. Ordered delivery is typically provided at transport or application layer.

2.3.6 Reliable Delivery

A service with reliable delivery guarantees that all data handed over to the protocol at the sender is delivered without corruption to the upper layer protocol at the receiver. If the delivery fails the reliable service typically notifies the sender of that failure whereas an unreliable service typically does not notify the other endpoint if delivery fails. To notify the sender of a successful transmission the receiver sends an *Acknowledgment (ACK)* so that the sender can mark the packet as acknowledged (further denoted as acked). If a packet does not get acked, the sender starts a *retransmission (RTX)* of that packet. A reliable service requires bidirectional connection and an unreliable one does not. These are exemplary mechanisms and others exist. An exhaustive explanation of mechanisms to ensure that no packet is lost and various retransmission strategies can be found in [4].

Various protocols on different layers provide reliable delivery. At link layer reliable transport is available on some links along the path, whereas on transport and application layer it may cover parts of, or the entire connection between the endpoints [15].

2.3.7 Handling of Data Corruption

During the transfer of a packet from the sender to the receiver bits within the packet can be corrupted through accidental or intentional modification in intermediate nodes or links. In order to detect a corruption typically additional data is added to mathematically verify its correctness. This added data can for example be a parity bit that simply indicates an odd or even number of zero bits; a checksum that is the computation of the sum of the data in a packet or a Cyclic Redundancy Check (CRC) [21] resulting from a polynomial division of the data contained in the packet. [15]

To detect potential deficiencies, the reverse operation is performed at the receiver which results in an error if the data is not correct. In this case the packet is dropped causing and in case of a reliable service, the sender typically retransmits the data. Detection of data corruption is offered by almost all layers.

There are alternative methods to detect data corruption that also provide additional functionality. Forward Error Correction (FEC) as explained in more detail in Section 3.5.2 can also be used to correct the data to a certain degree. Message authentication codes (MACs) are generated by computing a cryptographic function over the data. The receiver can use the MAC to cryptographically verify the identity of the sender.

2.3.8 Flow Control

When a fast sender transfers data to a slow receiver and the receiver cannot handle the data at that rate, the receiver may become overloaded. The flow control mechanism limits the sending rate of the sender to prevent the receiver from overloading, This is for example done by informing the sender with every ACK of the available buffer space in the so-called Advertised Receiver Window (RWND). Flow control is typically offered by specific protocols at transport layer.

2.3.9 Congestion Control

Congestion in a network happens if packets are buffered at bottleneck links or hosts along the path. The fuller the buffer is, the slower the packets traverse the bottleneck. As a result, the throughput of all connections with a path crossing the bottleneck is degraded. Congestion control is a mechanism to prevent this phenomenon and provides fairness by attempting to assign each connection an equal share of the link capacity.

This is done by regulating the sending rate of the sender. [15] describes the impact of the fairness of congestion control as: "*Congestion control is not so much a service provided to the invoking application as it is a service for the Internet as a whole, a service for the general good.*" There are various types of congestion control. [4] gives an in-depth explanation of various mechanism and types of congestion control. Congestion control is typically a service at transport layer.

2.4 Path Characteristics

The relevant path characteristics are packet delay, packet delay variation, Packet Loss Rate (PLR) and the product of the delay and the link capacity as the Bandwidth Delay Product (BDP).

2.4.1 Packet Delay

As a packet propagates from the sender to the receiver, it passes through a series of routers. Between each of these nodes (host or router) the packet suffers from several types of delays. The *end-to-end delay* δ_e can for example be split into: *processing delay* δ_c, *queuing delay* δ_q, *transmission delay* δ_t, and *propagation delay* δ_p [15]. Delay is often also referred to as *latency*.

Processing delay δ_c in a router is the time required to process the packet, for example by examining the packet header, checking bit-level errors and choosing a routing destination. When processing on application layer is included δ_c also includes processing of the incoming and outgoing data. This may include a multimedia coding, packetization, cryptographic operations, database lookups and so on. These delays typically only happen at the endpoints of the transmission and require further in depth application specific analysis which is important for time critical applications but not in the primary scope of this work.

The *queuing delay* δ_q indicates the time a packet is stored in a node waiting for transmission onto the link. When the traffic is low on the network, the packet may be passed through immediately which results in $\delta_q = 0$. When traffic is high, many earlier arrived packets may wait in the queue already which causes a large δ_q. In modern networks due to low memory prices nodes tend to have large buffers causing very high δ_q. Various research groups investigate this phenomenon call *bufferbloat* [22]. In multimedia streaming applications the receiver uses a so-called jitter buffer, which will

be explained in more detail in Section 2.6. Due to the nature of this buffer a large δ_q is added to the path.

The *transmission delay* δ_t is the time needed to transmit the data V over the link. This delay δ_t is limited by the available link capacity Γ of the path.

$$\delta_t = \frac{V}{\Gamma} \tag{2}$$

The *Propagation delay* δ_p defines the time it takes for a hypothetical 0-bit packet to propagate from one node to the subsequent node. This delay is defined by the propagation speed of the physical medium of the link. Measuring one-way delay requires the endpoints to have their clocks synchronized to have an identical view of time. A standardized procedure to measure the one-way delay can be found in [23].

The *Round Trip Time (RTT)* is the two-way delay between two endpoints. This characteristic is commonly known because it is easier to measure than the one-way delay. The RTT is continuously measured in many protocols and is therefore available to the application. To measure the RTT no time synchronization is needed. The path from the sender to the receiver may be different from the path from the receiver back to the sender. These are called asymmetric paths. Therefore, RTT measurements actually measure the performance of two distinct paths together [23]. The measured RTT also depends on the packet size V, the δ_c at the receiver, current traffic on the network and other factors. For simplicity these influences are ignored in the following and RTT computed as twice the end-to-end delay.

$$RTT = 2 \cdot \delta_e \tag{3}$$

The link capacity Γ, as shown in Equation (4) can be derived from the window size $CWND$ used for congestion control as described in Section 3.7.2. [1]

$$\Gamma = \frac{CWND}{RTT} \tag{4}$$

[1]For simplicity this formula does not consider packet loss on the path. A more precise formula can be found in [24].

2.4.2 Packet Delay Variation

Packet Delay Variation (PDV) ν, also called *jitter* is defined in [25] as the singleton ν_x of the difference between the one-way delays δ_e of a pair of packets in a given stream. Comparing a set of ν_x against the mean one-way delay $\overline{\delta_e}$ allows the calculation of the mean PDV $\overline{\nu}$ and statical values like its standard deviation $\sigma\nu$.

Some protocols measure the RTT with every packet that gets acknowledged. With this new measurement of the RTT in round i RTT_i, the sender updates its internal Smoothed Round Trip Time ($SRTT_i$) and Round Trip Time Variation ($RTTVAR_i$) variables. The SRTT is calculated as shown in Equation (5) as an approximation of $\overline{RTT}$. RTT_i is compared to the stored $SRTT_{i-1}$ to update the νRTT_i as shown in Equation (6). The recommended values for the smoothing factors given in [26] are $\alpha = \frac{1}{8}$ and $\beta = \frac{1}{4}$.

$$SRTT_i = (1 - \alpha) \cdot SRTT_{i-1} + \alpha \cdot RTT_i \tag{5}$$

$$\nu RTT_i = (1 - \beta) \cdot \nu RTT_{i-1} + \beta \cdot |SRTT_i - RTT_i| \tag{6}$$

2.4.3 Bandwidth Delay Product

The BDP refers to the product of the available link capacity Γ, also typically called the bandwidth of the path and the delay associated to that link in form of the RTT. The BDP refers to the amount of data in flight, the link can carry at any given time.

$$BDP = \Gamma \cdot RTT \tag{7}$$

2.4.4 Packet Loss

A packet loss describes the event that a packet was sent from the sender but does not arrive at the receiver correctly. A metric for packet loss is described in [27]. If a packet arrives but is corrupted by a bit error, it counts as lost. It is common to describe the loss characteristic of a path with the *Packet Loss Rate* (ρ) as a ratio between lost and all sent packets.

Along the path the packet passes several queues with limited buffer capacity. If a packet arrives at a full queue, this queue typically drops incoming packets [28]. Packet loss usually occurs in bursts of multiple consecutive packets.

Some queuing strategies that for example apply Differentiated Services (DiffServ) [29] do not drop the incoming packets but another packet in the queue with lower assigned priority. When the queue applies Random Early Detection (RED) [30], it computes a random probability to drop the incoming packet according to the fill level of the queue. This packet drops are proven useful to prevent network congestion when loss-based congestion control is used.

2.5 Dependability

Dependability is defined in [28] as: "*the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers*".

Events in a system with regards to dependability are defined in [28] as: "*A system failure occurs when the delivered service no loner complies with the specification, the latter being an agreed description of the system's expected function and/or service. An error is that port of the systems state which is liable to lead to subsequent failure: an error affecting the service is an indication that a failure occurs or has occurred. The adjudged or hypothesized cause of an error is a fault*".

The author in [31] classifies there faults and failures as: "*The temporal characteristics of faults vary widely, but can be roughly categorized as permanent, intermittent, or transient [. . .] Failures that prevent a component from functioning until repaired or replaced, such as the destruction of a network fiber by a backhoe, are considered permanent. Failures that allow a component to function properly some of the time are called intermittent. Damaged connectors and electrical components sometimes produce intermittent faults, The last category, transient faults, is usually the easiest to handle. Transient faults range from changes in the contents of computer memory due to cosmic rays, to bit errors due to thermal noise in a demodulator, and are typically infrequent and unpredictable.*"

Fault tolerance aims to tolerate faults by returning to correct service delivery after the occurrence of a failure and fault avoidance aims to prevent the effects of faults before the occurrence of a failure.

To tolerate faults additional information or components are typically added to the system. This so-called redundancy is defined in [31] as: "*Redundancy takes two forms,*

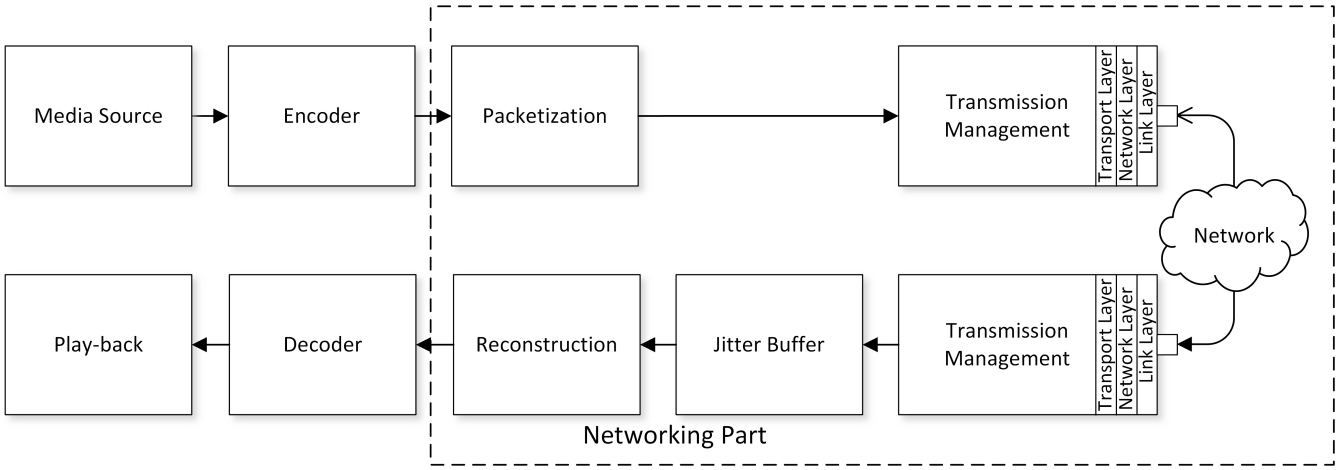

Figure 3: Components in a Streaming Application

spatial and temporal. Spatial redundancy replicates the components or data in a system. Transmission over multiple paths through a network and the use of error-correction codes are examples of spatial redundancy. Temporal redundancy underlies automatic repeat request (ARQ) algorithms, such as the sliding window abstraction used to support reliable transmission in the Internet's Transmission Control Protocol (TCP). A reliable network typically provides both spatial and temporal redundancy to tolerate faults with differing temporal persistence. Spatial redundancy is necessary to overcome permanent failures in physical components, while temporal redundancy requires fewer resources and is thus preferable when dealing with transient errors."

2.6 Real-Time Multimedia Streaming

Multimedia streaming describes the process of transferring audio and/or video data over a packet switched network to play back the media on a remote location. Multimedia streaming services are typically distinguished by the type of the media source. In case of a stored media source, the requirements for the system are different to the requirements when a live media source is used. The latter mode of operation is referred to as *real-time multimedia streaming* and is primary focus of this thesis. If a stored media source is used, the receiver can prefetch data to mitigate the effects of network fluctuating network characteristics [32]. Data cannot be prefeched in real-time multimedia streaming because that would require looking into the future. The challenges of real-time multimedia streaming are described in more detail in Section 3.2.

A typical structure and relevant components of a real-time multimedia streaming system is shown in Figure 3. It shows the path of the data traversing from the media source to the media play-back. The media source generates raw data in form of single pictures, the so-called *frames*. These frame are encoded to compress the size and save storage to reduce required link capacity for streaming. In the packetization component the encoded data gets prepared for transmission by adding the RTP header and slicing up the frame if the size is bigger than the MTU. The transmission management component handles the establishment and maintenance of the connection and passes the data down the network stack until the packet is transferred physically over the network. At the receiver, the packet gets passed up the network stack and stored in the jitter buffer. In the jitter buffer the packets are stored according to their timestamp and sequence number in their RTP header until passing them on to allow smooth play-back. In the reconstruction component the sliced frame is reassembled and the RTP header removed. The decoder transforms the encoded and compressed frame into a raw frame format, ready for play-back.

2.6.1 Video Codec

In order to save storage and transfer volume advanced video codecs are used to compress the video stream for more efficient storage and transfer over the network. Instead of all frames being compressed individually, typical video codecs encode some frames relative to the previous and subsequent frames. Typical video codecs like H.265 [33] and VP9 [34] use the following three frame types.

- **Reference Frames** — **I**ntra-Coded, I-frames — These frames serve as references for the encoding of other frames. If a reference frame is lost, all subsequent frames, until the next reference frame cannot be decoded and are therefore lost as well. Reference frames are significantly larger in size and have highest priority.
- **Differential Frames** — **P**redicted, P-frames — These frames encode only the difference to a previous reference or differential frame. Differential frames are smaller in size and when lost, the subsequent ones can still be decoded. Differential frames have lower priority.
- **Bidirectional Differential Frames** — B-frames — These frames use both previous and subsequent frames as a reference for encoding. They require the least size, and also have the least priority. Using subsequent frames as a reference, additional delay is required, which makes them not practical for real-time video streaming.

An in-depth analysis of the delay characteristics of video codecs with focus on real-time video streaming can be found in [35]. It shows that *intra-refresh frame coding*, which is not explained further in this thesis is the best choice for low-delay real-time video streaming.

2.6.2 Multimedia Streaming Related Protocols

The multimedia streaming protocols described in this section reside on application layer. RTP is used to transfer the actual media data and Real-Time Transport Control Protocol (RTCP) for signaling and a reverse channel from the receiver to the sender. Both are standardized in [36]. Other protocols for session establishment and negotiation are amongst others, Real-Time Streaming Protocol (RTSP) [37], Session Initiation Protocol (SIP) [38] and Session Description Protocol (SDP) [39].

Real-Time Transport Protocol

The Real-Time Transport Protocol (RTP) has first been standardized in 1996 as a protocol to transport multimedia data over IP networks and has become the primary standard for this purpose. It provides an end-to-end delivery service of data with real-time characteristics [40]. RTP does not guarantee the timely delivery of the packets and does not prevent out-of-order delivery but supports higher-layer applications in detecting packet losses or packet reordering. The transport characteristics are defined by the underlying protocol. In multimedia streaming the loss of packets can be tolerated in favor of timeliness. This is why typically UDP, as described in Section 3.7.1 is used as a transport layer protocol. For voice communication many concealment algorithms are available [41], which improve the subjective quality of the transmitted data as shown in [42].

RTP adds the following relevant information in its header to help the receiver reconstruct the media stream.

- **Payload type** — Identifies the media codec used on the payload of the packet. Payload identifiers are standardized and can be extended.
- **Sequence number** — The source assigns continuous sequence numbers to packets. This sequence number is used by the receiver to detect packet loss or out-of-order arrival.

- **Timestamp** — Indicates the relative time at the media source. The timestamp allows the receiver to calculate transport delay variation.

The receiver must implement techniques to reassemble and replay the media stream correctly. Upon arrival of the packet, the receiver stores the packet in a jitter buffer, which adds an artificially delay to the playback. Packets that arrive out-of-order can be rearranged in the buffer and passed on to the application in the correct sequence. The required delay to minimize packet loss can be calculated from the transport jitter, which can for example be measured using the timestamp values of the packets.

Real-Time Transport Control Protocol

RTP based on UDP only offers unidirectional transmission and is lacking any kind of congestion control. In order to provide feedback about the quality of the transmission to the sender, an additional channel from the receiver to the sender is needed.

This purpose is served by the second part of the RTP family, the Real-Time Transport Control Protocol (RTCP). The sender and receiver periodically transmit to the peer their gathered information about the quality of the data using RTCP. The sender can then adapt the data sent to the conditions of the network, for example by changing to a lower bandwidth codec.

3 Requirements Analysis

The requirements considered in this thesis are tripartite. Firstly, the physical environment surrounding the application is described along with abstractions that apply. Secondly, possible candidates are listed along with a summary of their benefits and downsides. Lastly, the section introduces the key performance metrics that form the basis of the evaluation.

3.1 Environment

The system investigated is a mobile, autonomous tram that aims to transfer multimedia data to a central station. The tram is connected to the central coordination station by the use of mobile cellular data connections like LTE [1], HSPA [2] and if available in certain areas, WiFi [3] as access network.

In such a setup, the characteristics of the wireless connections are highly dynamic. They change frequently while the tram drives through the city. The reception and thus the connectivity of the path changes or may fail when turning around a corner or entering a reception hole like when entering an underground section. Temporary high packet loss rates on the wireless access network are to be expected. When a handover occurs by the mobile data provider between two LTE radio stations or between LTE and HSPA, the IP address of the interface may change, causing the established connections to fail. Recovery from link failures is expensive in terms of resources and causes large delays and therefore should be avoided.

To narrow the scope of investigation, the content and characteristics of the data transferred is investigated in the following.

3.2 Real-Time Streaming Requirements

A video stream that is streamed in real-time should be played back with as little delay as possible. A common challenge is the PDV, the so-called jitter. To compensate delay variation on the network path the receiver uses a jitter buffer. In this buffer incoming packets are reordered and delayed according to a sequence number delivered with the streaming control data [43]. The jitter buffer passes on the data without jitter to allow smooth media playback. The additional information required by the jitter buffer is transferred by the use of RTP. Multimedia streaming applications are designed

to tolerate the loss of data within reasonable limits. It is preferable to occasionally lose a frame, causing a quality degradation over delaying the playback by waiting for retransmission of lost frames.

The investigations regarding multimedia in this thesis focus on video streaming and leaves aspects related to audio media aside. Video streaming data typically has high bit rate and has high demands for link capacity. To reduce the requirements for link capacity the video transferred is typically compressed and encoded using an advanced video codec.

Delay Requirements

Quality requirements for multimedia streaming are subject to standardization. The standards describe media codec related quality requirements. This thesis focuses on network induced delays so only delay-related requirements are presented.

ISO 16505 [44] for vehicle monitor systems specifies the requirement for the end-to-end system delay with 200 ms from video capture to video display. This limit is given for vehicle internal video system where the network and its properties are known and part of the development. The characteristics of the Internet are far less controllable than in the setup proposed by this standard.

The International Telecommunication Union (ITU) specifies delay requirements based on an objective quality measurement for voice communication systems. A graphical representation of their E-Model from [45] is shown in Figure 4. For voice, mouth-to-ear delay lower than 150 ms is not perceived as annoying by a human listener. Delays between 150 ms and 450 ms are acceptable and if the delay exceeds 450 ms the results are perceived as dissatisfying. These values given for voice communication also apply for real-time (conferencing) use of video and audio combined [15].

The unidirectional tolerable network delay is denoted as the *deadline* (D) in the following.

3.3 Connectivity

One main aspect with regards to the network requirements between the sender and the receiver is the *connectivity*. The IP Performance Metrics (IPPM) defines a connectivity metric in [47], such that if a packet can be transmitted from a sender to a receiver and its ACK can be transmitted back, connectivity between those endpoints exist.

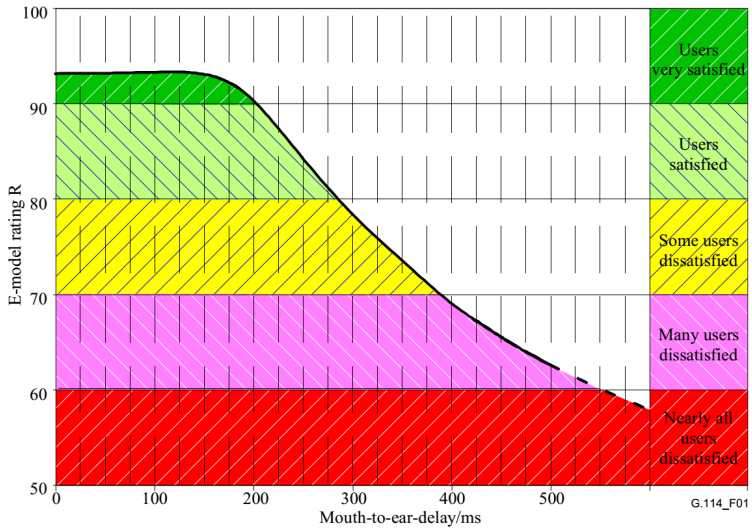

Figure 4: Determination of the effects of absolute delay by the E-model. (Source: [46])

The requirements with regards to data connectivity are high for the proposed setup. The provided service should ideally never be interrupted from an end-user perspective.

This metric is not sufficient for the proposed system because of the lack of time constraints in the metric. Therefore, the extended definition of connectivity given in [48] will be used. This also considers the timely constraints for the delivery of a packet. It defines a successful delivery if a packet reaches the receiver within the given deadline. "*Connectivity is the ratio of successful sends to total message sends over the time period of interest. [...] The metric is defined so that successful delivery of redundant packets does not improve the measured connectivity. Similarly, lost packets do not count against connectivity if the overall time deadline for the message is met through retransmission or redundancy.*" [48]

3.4 Fault Model

The so-called fault model is essential for building safety and time critical systems. "*Designing any system to tolerate faults first requires the selection of a fault model, a set of possible failure scenarios along with an understanding of the frequency, duration, and impact of each scenario. A simple fault model merely lists the set of faults to be considered; inclusion in the set is decided based on a combination of expected frequency, impact on the system, and feasibility or cost of providing protection.*" [31]

Event	Consequence	Temporal Ch.	Severity	Frequency
Bit error	Packet lost	transient	medium	medium
Packet arrives after deadline	Video frame lost	transient	medium	medium
Data packet lost	Video frame lost	transient	medium	high
ACK lost	Unnecessary retransmission	transient	low	high
P-Frame lost	Image distortion	transient	low	high
I-Frame lost	Video Lost until next I-Frame	transient	high	medium
Missing radio reception	Transient path failure	transient	medium	medium
IP address change	Permanent path failure	permanent	high	medium
Transient path failure	Connection loss	transient	medium	low
Permanent path failure	Connection loss, termination	permanent	high	low

Table 2: Fault Model

The faults regarding the network considered in this thesis are presented in Table 2. The temporal characteristics in the table refer to the consequences of the event. A lost frame is a single event that has a temporal effect on the output which is for example restored with the next frame. A failure of a path that leads to a termination of the connection is considered permanent because an entire setup of the connection with long delays is required. Packets in real-time video streams have different priorities and should therefore be protected accordingly by appropriate redundancy measures. The different packet priorities require a direct control over the multihoming mechanism to allow differentiation within the stream. Types of redundancy considered for this setup will be investigated in the following. Other faults that are relevant for correct system operation but considered to be outside the scope of this thesis include: video source failure, codec error, insufficient processing power, CPU overload, physical hardware failure, insufficient link capacity, weather influencing the radio signal, user failure, etc.

One main outcome of the fault model is the importance of redundancy to provide adequate continuous performance.

3.5 Redundancy

To counteract the previously described faults fault tolerance must be implemented by using redundancy or error correction. Various types of redundancy are explained in the following and an appropriate set of strategies is chosen for further investigation later in this work.

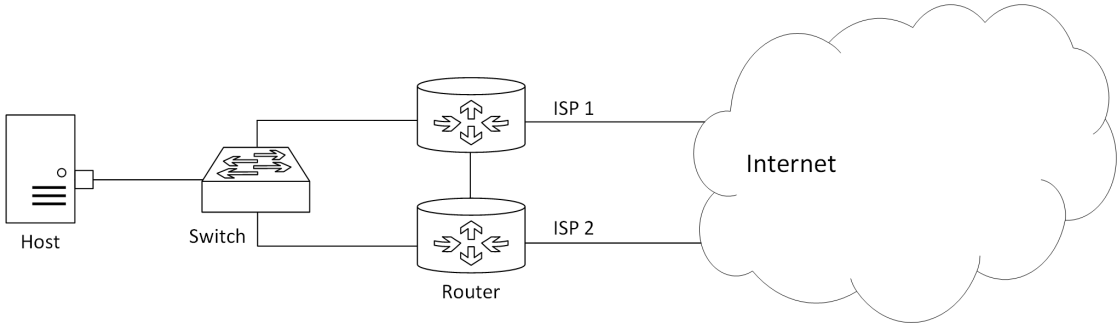

Figure 5: End-Site Multihoming

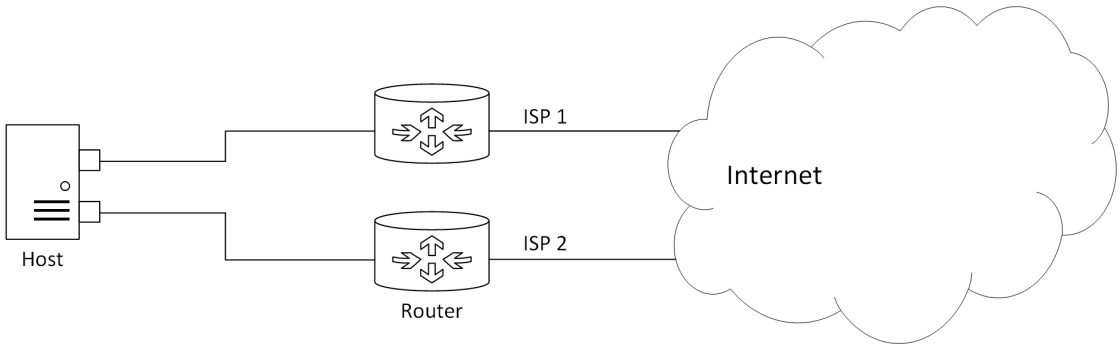

Figure 6: End-Host Multihoming

3.5.1 Multihoming

Multihoming is present when connecting a host or an entire network to more than one network. Multihoming may offer an increase in reliability as it allows connectivity in case of a connection failure to one of the connected networks. This practice is typically used to improve connectivity of enterprise networks or between networks of Internet Service Providers (ISPs).

The different types and characterization of multihoming are presented in the following. The two topology types as described in [49] are: *End-site multihoming*, when the network of a site is multihomed and *end-host multihoming*, when a host is multihomed directly. The classification in [50] distinguishes by the operation mode of the multihoming strategy. The multihomed connection can operate in *active-standby* or *active-active* connection mode where active-active is further classified if into *broadcasting* or *load sharing* operation. *Inter-flow* multihoming is present when path selection is done for each individual flow, whereas with *intra-flow* multihoming the path selection is done for individual packets.

End-Site Multihoming

End-site multihoming is present when multihoming is provided by the edge routers of the network of a site. A simple version of this is shown in Figure 5. It shows a single host that connects with a single link to a switch that forwards the packet according to the routing information and the multihoming protocol in use. The edge routers directly connect to the Internet by using two different ISPs. In general it is not visible to the endpoints of the network that the connection is multihomed. Typically, only inter-flow multihoming is available.

End-Host Multihoming

When the endpoint has multiple physical or logical network interfaces that directly connect to distinct access or core networks, this is called *end-host multihoming*. In this scenario the endpoint has knowledge of the status of the individual connections and can directly apply techniques and optimizations related to multihoming. Such a scenario is shown in Figure 6.

Active-Backup Operation

This type of operation requires the least complexity. It is provided when a path is the primary path and all other paths are backup. In a case of a path failure of the primary path, the outgoing packets are redirected to one of the backup paths. If the primary path fails permanently, another path is chosen as the new primary path. The procedure to move packets or the active status to a backup path is called *failover*.

To detect a path failure, for example sender-side timeouts are used. To satisfy the connectivity metric defined in Section 3.3, the failover procedure has to be achievable within the deadline as well.

Active-Active Broadcasting Operation

Active-active operation specifies that more than one path is active simultaneously. These can be all available paths or a subset.

The highest level of fault tolerance is provided by *active-active broadcasting*. With this type of operation data is duplicated onto the active paths. The link capability

of all paths needs to be sufficient to transfer the full data in time individually. The native throughput is not improved while the required link capacity is multiplied by the factor of $n - 1$, when n is the total number of paths. Packet loss on one path is theoretically not noticeable at the receiver. Active-active broadcasting does not suffer from failover delays and continues with full performance immediately if a least one path is operational.

Active-Active Load Sharing Operation

Active-active load sharing allows an aggregation of the available link capacity of all paths. A sender using this multihoming operation distributes data across the active paths. This can for example be done by alternating the destination for each packet in a round robin mechanism. More sophisticated queuing strategies exist that consider network state and cost of transfer. When load sharing is applied and one path fails, a failover delay has to be considered until the routing mechanisms detect the failure and initiate appropriate measures.

In [51], a technique to minimize the transportation cost by using multiple network interfaces is explained.

3.5.2 Forward Error Correction

Forward Error Correction (FEC) mechanisms improve the fault tolerance of the transmitted data against data corruption. It was first introduced in [52]. By adding additional redundant bits to the data, the receiver can detect or even correct bit errors, depending on the amount of redundancy added.

With this mechanism also entire redundant packets can be added to the stream to protect the transmission against packet loss. As described in [53], from a series of n packets, an additional redundant packet can be calculated by for example OR-ing the n original packets. If any one of the $n + 1$ packets is lost, the receiver can reconstruct the lost packet. If two or more packets are lost, the receiver cannot reconstruct them. If the number of packets n used for this operation is small, the level of fault tolerance is higher, but also introduces more transport overhead to the network therefore creating additional transmission delay. To reconstruct the lost packet the receiver has to wait for the entire group of packets to be received. [54] gives more details on FEC when used for multimedia streaming.

Another type of FEC are so-called fountain codes. One example is the raptor code, which is described in [55]: "*as many encoding symbols as needed can be generated by the encoder on-the-fly from the source symbols of a source block of data. The decoder is able to recover the source block from any set of encoding symbols only slightly more in number than the number of source symbols.*"

3.5.3 Interleaving

The data of the stream can be ordered so that the loss of a packet has less effect on the output quality of the media. Data of a recorded media section is split into multiple subsections and transferred in n packets [53]. If one of the n packets is lost, n sections are effected but for all only a fraction is lost. Especially for audio data, the quality degradation introduced by this loss is less than the loss of an entire section. The receiver has to wait for the entire n packets to play back the section, which adds undesirable delay.

3.5.4 Conclusion

Active-active broadcasting offers a good tolerance against path failure, although it requires significantly more data volume. The other multihoming types of operation, active-backup and active-active load sharing also tolerate path failure but add an undesirable failover delay for switching to another path. End-host multihoming gives the application or the protocol direct access to the multihoming functionality, which is required for intra-flow multihoming. End-site multihoming is provided by the edge routers where the application does typically not have direct control over the path selection. FEC mechanisms can correct data corruption and reconstruct lost packets. This technique is not directly related to multihoming, therefore it is considered out of the scope of this thesis.

3.6 Packet Replication For Multihoming

Redundancy through packet replication in a multihomed system can be used to increase the reliability. This section discusses various strategies to utilize the multihoming capabilities and replicate packets at various by the layers of the Internet architecture model.

3.6.1 Link Layer Approaches

For multipath transfer on link layer, multiple links can be bundled to appear as a single link to the network layer [4]. Bundling allows load sharing across the links, for example by splitting the data on the sender and reconstructing it on the receiver side or broadcasting the data to all links. Link bundling can only be used until the next network layer node, like a router. An implementation of link layer bundling is available in the Linux bonding module [56].

3.6.2 Network Layer Approaches

Some routing protocols on network layer that provide support for multihoming are presented in the following.

Multicast

Multicast routing provided by the IP can be used to deliver packets to multiple recipients, when using a multicast group as a destination address. To minimize the required overall link capacity multicast routing is optimized so that packets are duplicated on the last shared hop. The sender of a multicast message does not directly know the recipients of the multicast message. The communication is unidirectional and no reverse channel exists. Only unidirectional protocols like UDP can be used for multicast transport.

Intra-flow prioritization of packets is not possible. Path selection is done by the network nodes, responsible for the multicast routing and the sender has no direct influence on it.

Router Based Strategies

Router based technologies like Hot Standby Router Protocol (HSRP), Virtual Router Redundancy Protocol (VRRP) and Gateway Load Balancing Protocol (GLBP) allow multihoming and load sharing on network layer for end-site multihoming. By providing a virtual router, new clients are associated with one of the paths. Equal Cost Multi Path (ECMP)[57] provides inter-flow load sharing. Routing decisions are made for each new flow, like a new TCP connection.

LISP

LISP[58] provides location independent addressing. It adds a shim layer between IP addresses on network layer and applications on top of it. The identifier refers to a client whereas the locator refers to its network location in form of one or multiple IP addresses. The mapping between an identifier and the locator is done by a distributed database similar to Domain Name System (DNS). LISP provides load sharing for multihomed hosts by replying different locators to the LISP lookup request. An up-to-date survey of Locator/Identifier split approaches can be found in [59].

3.6.3 Transport Layer Approaches

[4] shows that intra-flow load sharing is best done on transport layer. Furthermore, decisions about individual packets based on their priorities are possible on transport layer because the protocols have direct control over the paths used for each packet. Section 3.7 investigates various transport layer protocols for their multihoming capabilities.

3.6.4 Application Layer Approaches

If the existing underlying protocols do not offer the required features, protocols on application layer may provide the required functionality specifically adapted to the use case. One example is Quick UDP Internet Connection (QUIC) [60], which aims to improve the overall performance for HTTP traffic. Amongst other features, it implements congestion control on top of UDP and offers faster connection establishment. This also means, moving the functionality into user-, instead of kernel space, which offers faster deployment.

3.6.5 Conclusion

Transport layer approaches offer the most potential to provide intra-flow prioritization and multihoming. For data connections over the Internet, link layer approaches are not applicable because link layer connections do not span between the endpoints. On network layer, protocols offering inter-flow multihoming functionality exist which cannot be used for the investigated system. Approaches on application layer are theoretically possible as well although many of the functionalities provided by the underlying protocols would have to be reimplemented on application layer.

3.7 Transport Layer Protocols

Transport layer protocols offer the highest potential and are therefore investigated in-detail in this section. Particular focus is on the evaluation of their capabilities regarding multihoming are shown.

3.7.1 User Datagram Protocol

UDP as defined in [61] provides an unreliable, connectionless, unidirectional service to the application. UDP is lightweight and therefore an application that uses it almost directly communicates with the network layer protocol [15]. The UDP header is 8 bytes long. UDP does not provide congestion control, which allows a sender to send at the maximum transmission rate of the link. This may lead to starvation of other connections or congestion in the network. Lost packets in UDP cannot be detected and retransmissions are not possible. For real-time multimedia streaming, the loss of data is acceptable, that is why UDP in combination with RTP is typically used for such a setup.

3.7.2 Transmission Control Protocol

TCP [62] provides reliable, ordered and error-checked delivery. The communication is bi-directional and connection-oriented. One type of packet replication is the retransmission mechanism of protocols with reliable delivery. TCP offers congestion and flow control to limit the sending rate according to the congestion state in the network and the processing capabilities of the peer. Various versions of TCP exist that implement different retransmission and congestion control strategies which are compared in [63].

A simplified version of the TCP congestion control as described in [65] defines four congestion control algorithms which are used in the different phases of congestion control: *slow start*, *congestion avoidance*, *fast retransmit*, and *fast recovery*. The values of the RWND and Congestion Window (CWND) variables defines the current rate limit for the sender.

CWND is the variable for the congestion control and represents the amount of data the sender can transmit before receiving an ACK. The RWND variable is given by the flow control and represents the amount of missing data the receiver can currently accept. Initially, a TCP sender is in the *slow start* phase and its CWND is set to one Sender Maximum Segment Size (SMSS), which is the MTU for TCP. In the slow start phase,

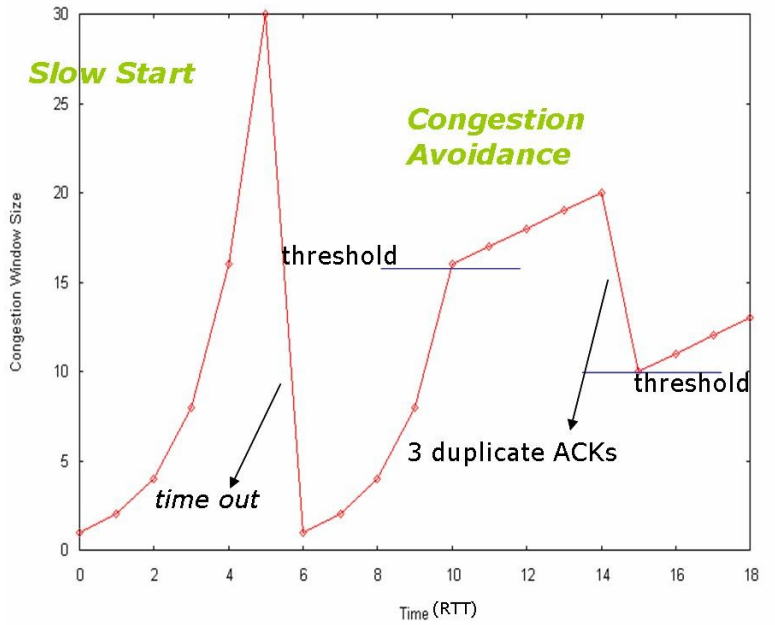

Figure 7: Congestion Window Trace [64]

CWND is incremented by at most one SMSS for each ACK received until CWND is larger than slow start threshold (ssthresh) and the sender transitions into *congestion avoidance*. [2] Whenever CWND is lower than ssthresh, slow start and when CWND is larger than ssthresh, congestion avoidance is used. In congestion avoidance CWND is incremented by at most one SMSS per RTT. If a chunk is not acked within a timeout (t_{RTO}), a Retransmission Timeout (RTO) is triggered causing the retransmission of that packet. After a RTO, the sender enters the slow start phase again and CWND is set to one SMSS, and ssthresh is set to $^{CWND}/_{2}$. When the sender receives three duplicate ACKs, the sender enters the *fast recovery* phase, triggering a *Fast Retransmission*. ssthresh and CWND are set to $^{CWND}/_{2}$. The sender stays in fast recovery phase until a non-duplicate ACK is received. In this phase the sender increments CWND by one SMSS for each addition duplicate ACK. "*This artificially inflates the congestion window in order to reflect the additional segment that has left the network.*" [65]

Figure 7 shows a trace of the CWND in red and the congestion control phases in green. The event of a RTO and the triple duplicate ACK are indicated with arrows and the ssthresh is shown as the threshold line.

[2] The initial value for ssthresh is set arbitrarily high to allow this value to be determined by the network.

Due to TCP offering reliable and ordered delivery, a lost or delayed packet may lead to *head-of-line* blocking causing undesired delays. Researchers in [32] show that when the maximum supported TCP throughput of the path is roughly twice the media bit rate, streaming over TCP results in minimal starvation and low buffering delays.

3.7.3 Datagram Congestion Control Protocol

Datagram Congestion Control Protocol (DCCP) [66] is an unreliable, message-oriented transport layer protocol which is intended for lightweight multimedia streaming. DCCP implements, amongst other features congestion control, lightweight connection establishment and variable length checksum fields in the header to minimize the overhead.

The standard specifies *TCP-like congestion control*, *TCP-friendly rate control* to throttle the transmission. TCP-friendly rate control is receiver-based and the receiver transmits ACKs less frequently to the sender to accumulate more information. DCCP does not specify if a lost packet shall be retransmitted or not and leaves this decision to the application. Researchers in [67] specify a Deadline-based Selective Retransmission strategy with the focus on low end-to-end delay. The application decides whether a packet shall be retransmitted based on a deadline, specified for each packet. This resulted in an overall improvement of the end-to-end delay.

3.7.4 Multipath TCP

Multipath TCP (MP-TCP) [68] extends TCP to transfer data on multiple paths simultaneously enabling intra-flow active-active multihoming with load sharing operation. A MP-TCP connection consists of multiple *subflows*, one for each underlying path. Each of these subflows appear as an autonomous TCP connection which allows minimal modifications on intermediate nodes and faster deployment and minimizes the influence on middle-boxes onto MP-TCP traffic.

Congestion control for the individual subflows is separated and out-of-order arrival of packets on MP-TCP level, caused by dissimilar path delays does not cause degradation of the overall throughput. Each of the subflow offers fully reliable transport. A packet that has been sent on one path cannot be removed from this subflow or retransmitted on another subflow. Active-active broadcasting operation is not feasible in MP-TCP because for example a packet that has been successfully delivered on one path but is lost on another, cannot be removed from the sending queue of the other path.

An extension to enable partial reliability for MP-TCP has been proposed in [69]. This extension allows already sent data on subflows to be abandoned similarly to the PR extension of SCTP explained in the following.

3.7.5 Stream Control Transmission Protocol

SCTP was first standardized in the year 2000 for the use in Signaling System 7 (SS7) for packet switched telephony routing and updated and extended since to allow other use cases. SCTP provides many of the features of TCP, like reliable ordered delivery and connection-oriented service and allows unreliable, unordered delivery like UDP for parts of the data. The standard offers loss-based congestion control like TCP and provides transport layer multihoming. SCTP can selectively transfer packets unordered and/or unreliable, while others in the same flow are transferred with ordered and reliable delivery.

PDUs in SCTP are called *chunks* which are numbered by the sender with a Transmission Sequence Number (TSN) to enable ordered delivery. Multiple chunks can be *bundled* into a single packet that is handed over to the network layer protocols to reduce overhead when sending chunks smaller than the MTU.

A connection between two SCTP *endpoints* is called an *association* which may contain multiple paths, identified by their network layer addresses. Addresses of an endpoint are specified during the association establishment. If multiple addresses are available, one is defined as primary and the others as backup paths. Addresses can be added or removed during the association lifetime with SCTP's Dynamic Address Reconfiguration (DAR) extension [70]. This can be used to implement mobility and [71] mentions this feature as a possible implementation of the locator-identifier split. SCTP can deliver multiple streams to different applications over a single association. The *stream identifier* and the *Stream Sequence Number (SSN)* in the chunk header are used to deliver chunks for each stream in-order. Ordering can be disabled for individual chunks when the U-Flag is set in the chunk header. The receiver immediately passes the chunk to the application and does not keep them in the receiver buffer.

The default congestion control used in SCTP is similar to the congestion control of TCP described in Section 3.7.2. To signal the reception of chunks the receiver sends a Selective Acknowledgment (SACK) chunk, which in addition to the Cumulative Acknowledgment (CumACK) point contains gap reports for chunks that have arrived out-of-order. The sender maintains individual queues and CWND variables for all configured paths. An SCTP receiver typically delays the sending of a SACK to improve link capacity utilization. The SACK Immediately extension [72] introduces the I-Flag to the

chunk header to inform the receiver that the SACK should be sent immediately. This is useful in a delay-sensitive system that sends bursts of data where for the last chunk in a burst sequence waiting for another chunk is not practical. Another throughput enhancing but delay increasing mechanism is Nagle's Algorithm which was first proposed in [73]. This algorithm intends to reduce the header overhead by only sending full MTU-sized packets. If the data ready for sending is less than the MTU, sending of this data is delayed until sufficient data is ready. This optimization is useful to reduce protocol overhead but additional delay is unfavorable for delay-sensitive services.

SCTP offers fragmentation where the sender splits incoming data into chunks to transmit MTU-sized packets. The receiver can only deliver the data to the application when all fragments are received successfully. In multimedia streaming partial data loss is typically acceptable which makes SCTP's fragmentation functionality not usable in this context. The multimedia data should instead be split up before passing it to SCTP.

With the Partial Reliability (PR) extension [74], the sender can *abandon* chunks if a timely deadline has passed or, as added in [75] a maximum number of retransmissions have occurred. To abandon chunks, the sender sends a *FORWARD-TSN* chunk, signaling which chunks have been abandoned.[76] shows that the FORWARD-TSN mechanism is insufficient when reliable and unreliable chunks are mixed in the same association and that the Non-Renegable Selective Acknowledgments (NR-SACK) mechanism, explained in Section 3.8.1 improves this situation.

To monitor the available paths, the RTT is measured on every arrival of a SACK and the $SRTT$, νRTT and t_{RTO} variables are calculated. t_{RTO} is calculated as shown in Equation (8).

$$t_{RTO} = SRTT + 4 \cdot \nu RTT \tag{8}$$

SCTP uses the RTO to detect path failures. With every occurrence of an RTO, t_{RTO} is doubled until a new SACK on that path is received and after a certain number of occurrences the path is marked as failed.[77] shows that this path failure detection mechanism may react slowly. The Quick Failover Algorithm extension [78] describes a mechanism to improve the performance of the path failure detection by defining a new Potentially Failed (PF) state for paths. After a configurable number of RTOs, usually 1, the effected path enters the PF state where no new chunks are sent and instead path *heartbeat probing* is enabled. On reception of a heartbeat or a data acknowledgment on a path in PF state, it is reactivated.

The Concurrent Multipath Transmission (CMT) extension enables active-active load sharing operation for SCTP by allowing the sender to transfer chunks on multiple paths

Service/Features	UDP	TCP	DCCP	SCTP	MP-TCP
Connection-Oriented	no	yes	yes	yes	yes
Reliable data transfer	no	yes	opt	yes	yes
Partially-Reliable data transfer	no	no	opt	yes	opt
Ordered data delivery	no	yes	no	yes	yes
Unordered delivery	yes	no	yes	yes	no
Flow control	no	yes	yes	yes	yes
Congestion Control	no	yes	yes	yes	yes
Selective Acknowledgments	no	opt	yes	yes	opt
Application PDU fragmentation	no	yes	yes	yes	yes
Application PDU bundling	no	yes	no	yes	yes
Multi Streaming	no	no	no	yes	no
Multihoming	no	no	no	yes	yes
Allows half-closed state	n/a	yes	yes	no	yes
Reachability check	no	opt	no	yes	opt

Table 3: Comparison of Transport Layer Protocol Features (Inspired by: [80])

simultaneously. It was first introduced in [79] and investigated in detail in [4]. The features added by CMT and the challenges with active-active load sharing operation on transport layer are investigated in Section 3.8.

3.7.6 Comparison and Conclusion

Table 3 summarizes the features of the transport layer protocols UDP, TCP, DCCP, SCTP and MP-TCP described in this section. [3] For the given requirements for real-time multimedia streaming in a multihomed setup, out of the investigated transport layer protocols, SCTP offers the most appropriate features, in particular with its CMT extension which is investigated in more detail in the following.

[3] Not all features mentioned in the table were explained in-detail for all protocols in this section. Some additional features are mentioned in the summary for completeness.

3.8 Concurrent Multipath Transmission

CMT extends SCTP to allow intra-flow active-active multihoming with load sharing operation. It extends standard SCTP amongst others by the following features targeting specific challenges that are found in active-active load sharing operation on transport layer: split fast retransmissions, appropriate congestion window growth and appropriate delaying of SACKs. These are explained with illustrative examples in [81].

If the path delays in a multihomed system are dissimilar, the receiver receives many packets out-of-order. These are acked in the gap report in a SACK. The congestion control in SCTP increases CWND only when a new CumACK is received. When the data is cumulatively acked later, only the new cumulatively acked chunks contribute to the growth of CWND and chunks acked previously by a gap report are ignored. A CMT sender correctly adapts the CWND by maintaining a virtual CumACK per path, as explained in [79]. The frequent gap reports also lead to unnecessary Fast Retransmission because the sender may receive many SACKs with identical CumACK [82]. The sender therefore considers the transmission paths before triggering a Fast Retransmission. When the receiver in standard SCTP receives a chunk out-of-order, the SACK is sent immediately to inform the sender of the missing packet. A CMT receiver therefore maintains individual counters per path for delayed SACKs.

3.8.1 Buffer Blocking

For a sender to fully utilize the link capacity of a path it should always have as much data in flight as the link can transfer, which is indicated by the BDP. The sender, as well as the receiver buffer have to be able to store at least that much data. A shared buffer for all paths in a multihomed connection has to be able to store at least $\sum_i BDP_i$ [4], where BDP_i represents the BDP of the path with the index i. [83] proposes that the receiver buffer should be at least $(\sum_i \Gamma_i) \cdot RTT_{max}$, where RTT_{max} is the highest RTT of all considered paths. A path with $RTT = 50ms$ and $\Gamma = 30Mbit/s$ has $BDP = 187.5kbyte$. Typically buffers are assigned 64 kByte of memory. The required buffer space for multiple paths may be very large. It is therefore fair to assume that the buffer space is limited. The researchers in [4] describe in detail the different types of buffer blocking. Buffer blocking exists for services with ordered and unordered delivery. Types of buffer blocking that affect unordered delivery impair ordered delivery, but not vise-versa [4].

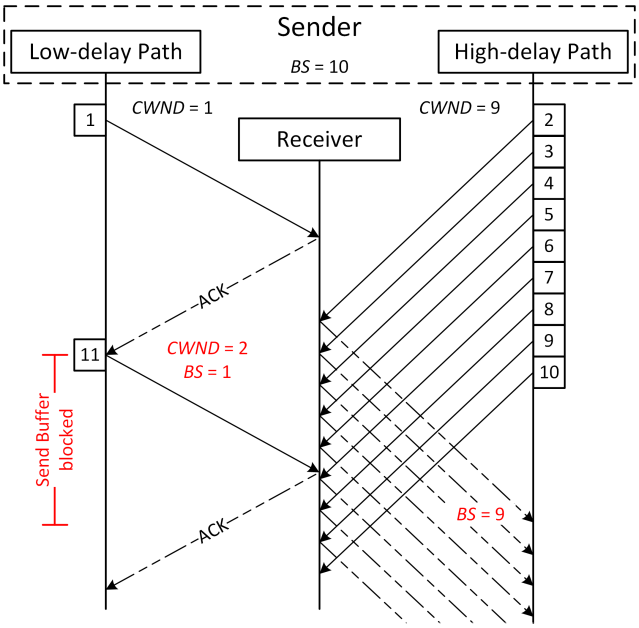

Figure 8: Transmission-Induced Sender Buffer Blocking

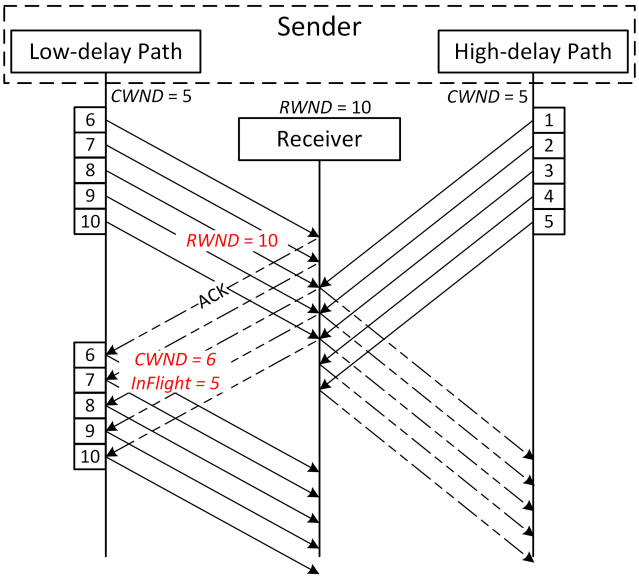

Figure 9: Advertised-Window-Induced Receiver Buffer Blocking

Transmission-Induced Sender Buffer Blocking

To illustrate *transmission-induced sender buffer blocking* an example is presented in Figure 8. The connection in the example consists of two paths. One has a high and the other a low end-to-end delay δ_e. The shared sender buffer can store 10 packets. The remaining shared buffer space at a given time is indicated by BS. $CWND$ for the low-delay path allows 1 packet and $CWND$ for the high-delay path allows 9 packets to be sent. The sender sends according to the allowed CWNDs on both paths. After the transmission of these packets, the remaining BS is 0. [4] The packet with sequence number #1 from the low-delay path is acked before the packets #2-#10 on the high-delay path are acked. #1 is removed from the sender buffer and CWND of the low-delay path is increased to 2. [5] Now there is only space for one packet ($BS = 1$) in the sender buffer and the sender cannot fully utilize the link capacity indicated by $CWND$.

This cause for buffer blocking is shown in [4] to be eliminated by buffer splitting as described above.

[4] The illustration highlights the transmission of the first packet and ACK, whereas the gray arrows indicate the transmission of the subsequent packets.

[5] All examples assume the sender is in congestion avoidance phase.

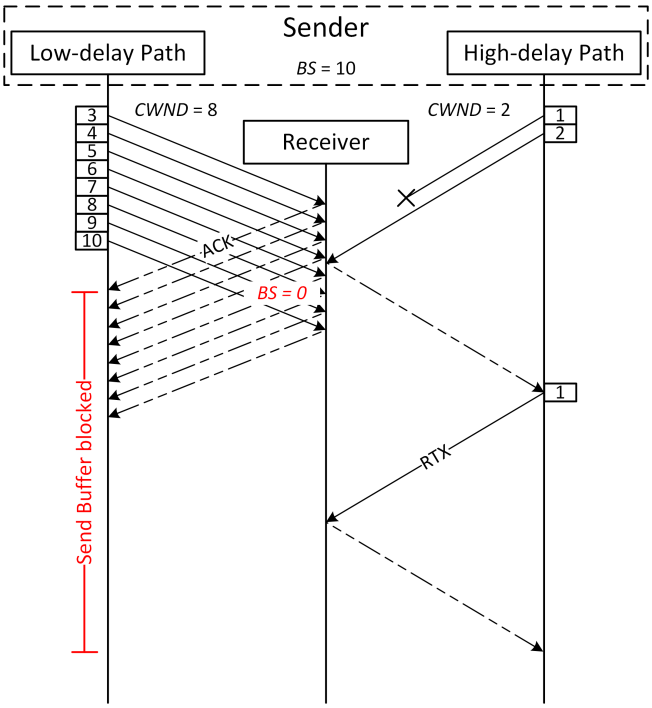

Figure 10: Reordering-Induced Sender Buffer Blocking

Advertised-Window-Induced Receiver Buffer Blocking

Advertised-window-induced receiver buffer blocking may appear in unordered delivery and therefore also in ordered delivery. The example shown in Figure 9 uses unordered delivery for all transferred packets. The shared receiver buffer can store 10 packets and informs the sender of this by the use of the RWND. The sender sends packets #1-#5 over a high-delay path and #6-#10 over a low-delay path. When #6-#10 arrive at the receiver, the packets are delivered to the application and the ACK from the receiver indicates again a RWND of 10. The sender increases its CWND upon reception of the ACK. With $RWND = 10$ and 5 packets in-flight, the sender is allowed to send 5 packets but the capacity of the path ($CWND = 6$) can not be utilized [4].

To reduce the effect of advertised-window-induced receiver buffer blocking, buffer splitting can be used.

Reordering-Induced Buffer Blocking

Reordering-induced buffer blocking exists for the sender buffer, as well as for the receiver buffer and is also known as *head of line blocking*. In the example presented in Figure 10 the sender buffer is blocked. The shared sender buffer can store 10 packets. CWND of the high-delay path allows 2 packets to be sent, whereas the CWND of the low-delay path allows 8 packets. Packets #1 and #2 are transmitted on the high-delay path and #3-#10 on the low-delay path. #2-#10 are successfully transmitted and #1 is lost. [6] The receiver might reneg one of the already received packets forcing the sender to keep the packets in the sender buffer. After the ACK for #3 was received the sender buffer is blocked. Because #1 and #2 are not acked at this point, the sender has to keep #3-#10 in the sender buffer for potential later reneging. The sender buffer is blocked until #1 is acked successfully. In the worst case #1 is lost again in the retransmission and the entire connection is blocked until the occurrence of a RTO [4].

With unordered delivery, the sender does not need to hold the data because it is non-renegable, and already delivered to the application. With the use of NR-SACK this situation and the sender buffer is unblocked.

Non-Renegable Selective Acknowledgments

NR-SACK [84, 85] is an alternative acknowledgment mechanism that extends the original SACK format to allow the sender to free non-renegable and therefore unneeded chunks from the sender buffer. When packets arrive out-of-order, the receiver of a service that provides ordered delivery, stores these packets in its incoming queue. Until the arrival or the abandonment of a chunk, all chunks in the sequence up to that chunk cannot be delivered to the application. When this so-called *head-of-line blocking* is present and the buffer fills up, the receiver may drop (*reneg*) chunks to make space for newly arriving chunks. Chunks that arrive out of order are implicitly renegable. This acked but renegable chunks have to be kept in the outgoing queue of the sender because if the receiver renegs such a chunk, the sender has to be able to retransmit it. When chunks in SCTP are transmitted with unordered delivery, they are passed to the application directly upon arrival and are non-renegable. In this case the sender can free the buffer space but the information delivered in the standard SACK chunk is insufficient [84]. The required information is added with the NR-SACK chunk.

[6] For simplicity the example assumes that the sender detects the loss of #1 already after the ACK of #2 and retransmits #1.

Buffer Splitting

When the path delays are dissimilar, a shared buffer eventually leads to unbalanced distribution of the data and buffer blocking [4].[86] introduces the splitting of shared buffers into equal shares for all involved paths. This concept is extended in [87], which takes into account that buffer space can be occupied in the sender buffer, even though a packet has already been acked. The splitting then considers the amount of missing bytes per path. A new packet is allowed to be sent if Equation (9) holds true. O_i represents the amount of bytes missing on path i. $V(p)$ is the amount of data to be sent by the packet p. BS is the amount of bytes the shared sender buffer can hold and n the number of paths involved.

$$O_i + V(p) \leq \frac{BS}{n} \tag{9}$$

[4] shows that buffer splitting based on the amount of missing bytes significantly increases the overall throughput, especially when sender and receiver buffer space is unequal. Furthermore, it helps against bufferbloat, where the path has a high queuing capacity that potentially exceeds the sender or receiver buffer space.

3.9 Conclusion

The fault model shows that protection against path failure is critical in the given environment. To protect the connection from path failure, multihoming on transport layer has the highest potential to improve the dependability of a system. Active-active broadcasting offers the highest protection against path failure when continuity of service is required. Non of the investigated protocols and mechanisms offer this mode of operation, although the transport layer protocol SCTP offers a variety of features, suitable for real-time multimedia streaming. In the following optimized active-active broadcasting operation with a focus on SCTP is developed.

4 Systems Model and Alternatives

This section describes in more detail different strategies for packet duplication. It starts with the already presented fast retransmission mechanism that is practical for the given application under certain circumstances, followed by description of a Full Packet Duplication (FPD) strategy that duplicates all packets onto multiple paths. The last strategy described is the newly introduced Deadline Based Preemptive Retransmission (DPR) that intends to gain benefits from both of the above strategies while still meeting the deadline requires. The descriptions are meant to be generic although with a focus on multimedia streaming and certain aspects of SCTP.

4.1 Key Performance Metrics

In order to evaluate and compare the performance of potential candidates it is necessary to define performance metrics, i.e., what to actually measure. The newly developed metrics are presented in the following.

For all metrics the connection is considered properly established and initialized. When running an experiment, as an example, the congestion window for protocols with congestion control must be sufficiently large. Furthermore, the state of the buffers along the path is of relevance. On a cold start of an experiment the buffers might be empty and influence the initial performance.

All metrics are examined in the interval $[\tau_0, \tau_E]$. At τ_0, the system is considered in a steady state whereas τ_E marks the end of the experiment with $\tau_E = \tau_0 + t_e$ and $t_e > 0$ is the time of the metric observation. The set of considered packets P contains the packets p_i with the index i. Corrupted packets are considered lost.

4.1.1 Time To Delivery

The TTD_i for p_i defines in an abstracted application the difference in time between the time the packet p_i enters the sender's packetization module at τ_{Si} and the time the packet leaves the receiver's packet reassembly module at τ_{Ri}.

$$TTD_i = \tau_{Ri} - \tau_{Si} \tag{10}$$

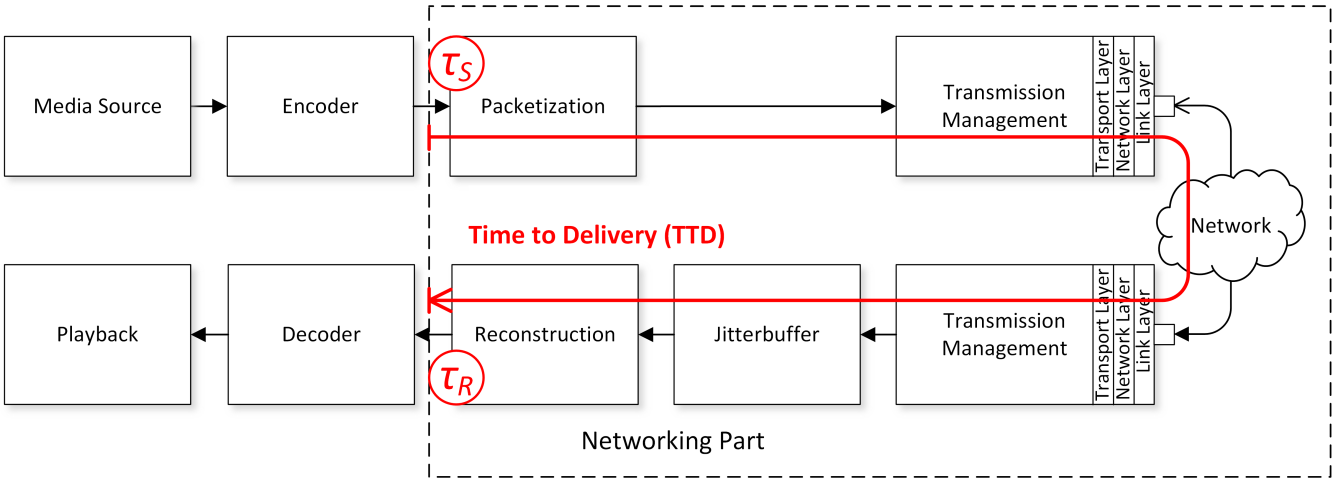

Figure 11: Measurement of Time To Delivery

Figure 11 shows how the Time To Delivery (TTD) is measured. The networking part includes the actual transmission over the network with all its buffering queues at the sender, receiver and all intermediate nodes. At the sender, the packetization and the transmission management and at the receiver, the transmission management, the jitter buffer and the reconstruction of the frame are included. Not included in the TTD, are: the capturing, encoding and compressing of the media input at the sender and the decoding and play-back of the media at the receiver.

4.1.2 Delivery Within Deadline Ratio

Delivery Within Deadline Ratio (DDR) is a metric that reflects connectivity as defined in Section 3.3 with strict constraints for the packet delivery time. A packet p_i is delivered within the deadline D, if $TTD_i \leq D$ for the first arrival of p_i. Duplicate arrivals of p_i are ignored. The DDR defined in Equation (11) defines the ratio between the number of packets delivered within the deadline — the number of elements in the set of packet, where for the packet p_i the TTD_i is lower or equal the D — and the number of all packets that entered the networking part of the transmission in the time interval of examination $[\tau_0, \tau_E]$ represented by the set P as defined in Equation (12).

$$DDR = \frac{|\{p_i : TTD_i \leq D\}|}{|P|} \tag{11}$$

$$P = \{p_i : \tau_0 \leq \tau_{Si} \leq \tau_E\} \tag{12}$$

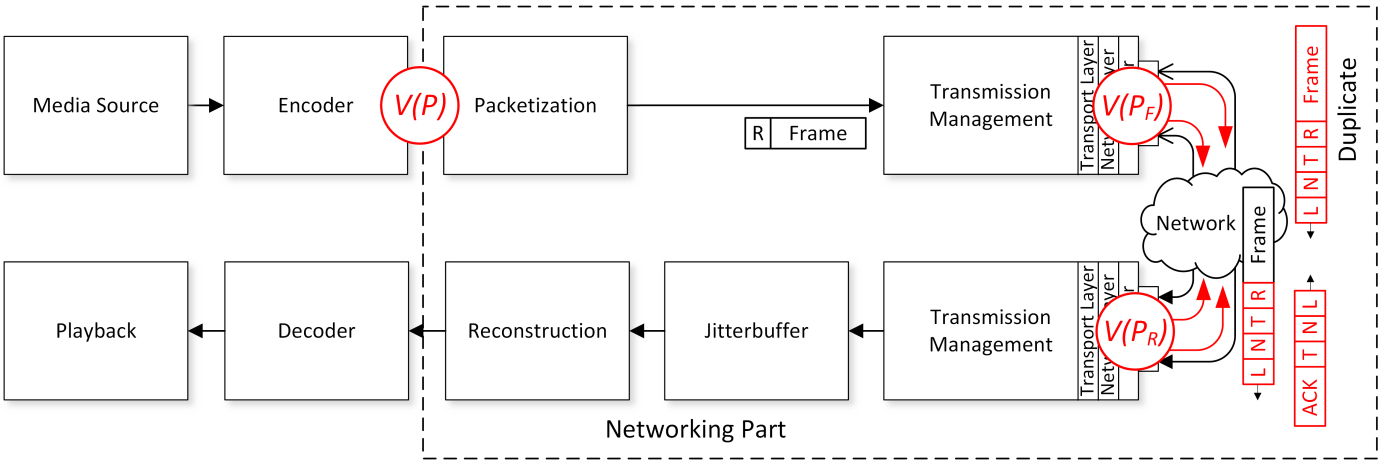

Figure 12: Measurement of Transmission Overhead

4.1.3 Path Failure Impact

The Path Failure Impact (PFI) defines the impact of a single permanent path failure to the connectivity of the system for a certain time of interest after the occurrence of the path failure. The aim of this metric is to show the performance of a duplication strategy to tolerate a path failure and provide uninterrupted delivery of packets.

To determine the impact of a path failure the DDR of a time interval of length t_m, starting at the time of path failure τ_{PF} is calculated as shown in Equation (13).

$$PFI = DDR(\{p_i : \tau_{PF} \leq \tau_{Si} \leq \tau_{PF} + t_m\}) \tag{13}$$

4.1.4 Transmission Overhead

In packet switched networks with a layered architecture as described in Section 2.2 additional data to the payload is sent over the network. The author of [88] describes this protocol overhead as the number of bytes in the headers used for protocol encapsulation. The payload is the actual data to be transferred. In the context of this thesis the payload describes the multimedia data as produced by the multimedia codec. In addition to the protocol overhead the Transmission Overhead (TO) includes the amount of data needed for redundancy mechanisms like packet duplication and retransmissions. This definition is chosen because mobile Internet providers charge by data volume sent on the network. TO indicates the cost for the provided reliability and protocol choice.

The function $V()$ over a set of packets results in the sum of bytes of all packets in the set. Packets in forward direction include protocol headers for transport, network and link layer as well as packet retransmissions or duplications if the protocol offers reliable service. In reverse direction, ACKs packets are included.

To calculate the TO the total size (volume) of multimedia packets in the set P, $V(P)$ is compared to the volume of the set of packets sent on the link P_L. P_L as shown in Equation (14) consists of the packets on N paths sent in forward direction (P_F) and the packets in reverse direction (P_R).

$$V(P_L) = \sum_{i}^{N} V(P_F^i) + \sum_{i}^{N} V(P_R^i) \tag{14}$$

Figure 12 shows in red the locations along the data path where the measurements of TO are done.

$$TO = \frac{V(P_L)}{V(P)} - 1 \tag{15}$$

4.2 Fast Retransmission

Fast Retransmission (FRTX) is the default mechanism to provide reliability through packet duplication in TCP and SCTP. A packet loss is detected by the cumulated acknowledgment (CumACK) information transmitted from the receiver to the sender. In this section it is assumed that the receiver sends an explicit SACK packet to the sender that includes the information required for the sender to detect a packet loss. A packet is typically fast-retransmitted after the CumACK information reports the same value three times. In CMT the CumACK information is considered for each path individually.

If a packet gets lost on the path, the receiver informs the sender with the SACK for the subsequent packets followed by the receiver retransmitting the packet. This Fast Retransmission procedure completes with the reception of the retransmitted packet after $\delta_R = 2\delta_e^S + \delta_e^R$ [7], where δ_e^S represents the one-way delay in the sender direction and δ_e^R the one-way delay in the reverse direction. When a symmetrical path is assumed, where $\delta_e = \delta_e^S = \delta_e^R$, the Fast Retransmission procedure completes after $\delta_R = 3\delta_e$. For

[7] This ignores the fact that packet loss in only detected after a triple-duplicate acknowledgment.

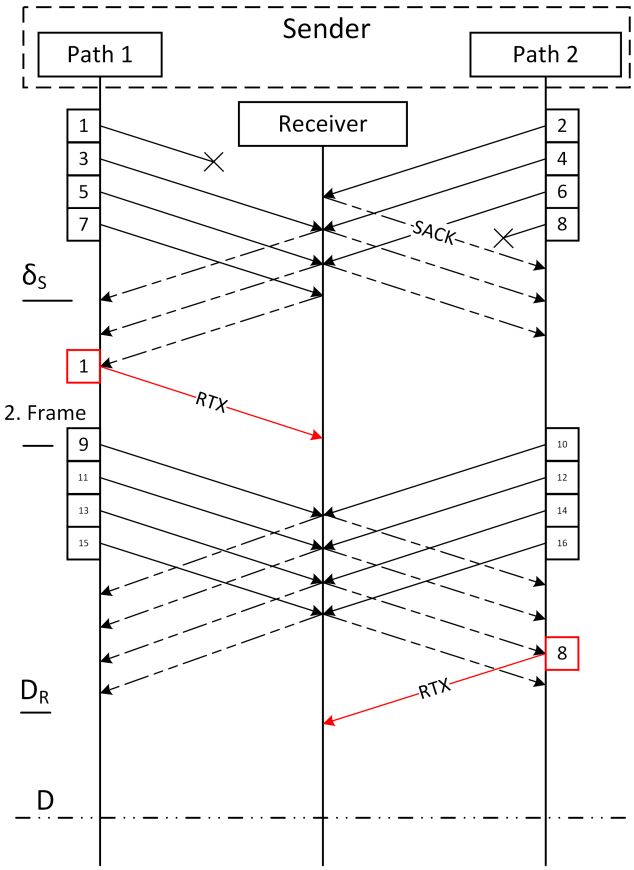

Figure 13: Packet duplication through Fast Retransmission

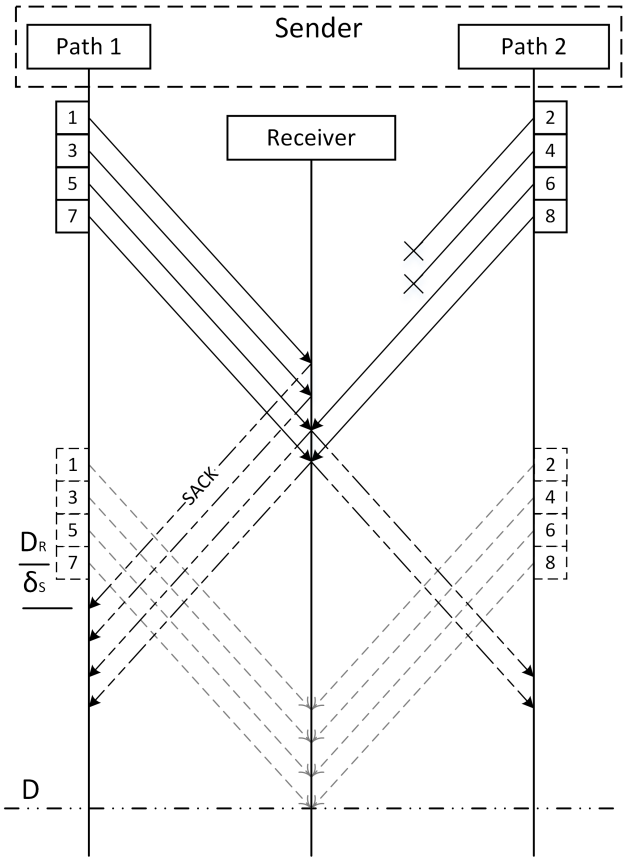

Figure 14: Fast Retransmission not working anymore

the fast retransmission mechanism the upper limit given in Equation (16) is a necessary condition to allow reliability through packet duplication. This condition is not sufficient as loss may occur again on a retransmission.

$$\delta_e \leq \frac{D}{3} \tag{16}$$

In a real-world scenario, δ_e is not static and depends amongst other influences on the packet size and the congestion state on the network. This variation of δ_e is typically measured regularly and represented by the PDV ν. In order to give an upper limit the maximum PDV $\widehat{\nu}$ has to be considered. In this thesis $\widehat{\nu}$ is not considered in the equations and evaluation.

Figure 13 shows an example of the fast retransmission mechanism. Packet #1 on path 1 is lost and detected on the reception of the third SACK on path 1 causing a fast-retransmission of #1, shown in red. The receiver in general returns the SACK on the same path as the data has been received. Although, this is not mandatory according to the specification. Especially when data from multiple paths was received, the receiver may bundle the SACK and return it on a single path. The standard does not specify a deterministic behavior for this scenario. In Figure 13 the SACK is drawn on the same path as the data because the paths are logically separated with regards to the CWND growth, CumACK and buffer space. Furthermore, the paths in this representation have symmetrical delay where the return path is irrelevant for to round-trip time. In the example the sender periodically transmits bursts of packets like generated in a video stream. The last packet of the frame (#8) on path 2 is lost but the SACK mechanism can only detect the loss if packets are transmitted afterwards on the same path. The next packet is transmitted with the next frame of the stream. With a frame rate of f, the next packet is transmitted earliest after a time of $^1/_f$ [8]. In this case the Fast Retransmission procedure completes after δ_R^L as shown in Equation (17). Equation (18) includes this into the upper limit for δ_e.

$$\delta_R^L \approx \frac{1}{f} + 3\delta_e \tag{17}$$

$$\delta_e \leq \frac{D - \frac{1}{f}}{3} \tag{18}$$

A relevant point in time for a successful transmission within the deadline is the so called retransmission deadline D_R. It represents the last moment a packet can be transmitted to arrive within the deadline D. It is presented in Equation (19).

$$D_R = D - \delta_e \tag{19}$$

In Figure 14 an example is shown where these boundaries are not met because the transmission is too slow so that the first SACK arrives even after D_R. The dotted packets after D_R indicate the theoretical last chance for a transmission at D_R. With this network condition the Fast Retransmission mechanism does not provide the desired reliability and a different approach as shown in the following sections is needed.

[8] It is here assumed that the queuing delay δ_q at the sender and the transmission delay δ_t of the frame are approximately 0.

4.3 Full Packet Duplication

In a multihomed system, Full Packet Duplication (FPD) is a strategy that offers active-active broadcasting on multiple paths. Instead of waiting for the acknowledgment to indicate the loss of a packet, all packets are transferred redundantly on multiple paths.

Standard multipath-enabled transport layer protocols, like SCTP and MP-TCP do not offer broadcasting. The CMT extension of SCTP is still experimental and only focuses on load sharing to utilize multiple paths in order to improve the overall throughput.

The scheduling approach for Full Packet Duplication used in the example shown in Figure 15 transmits the packets of the frame on all paths with a round-robin path selection. After the entire frame was scheduled the entire frame is scheduled again in round-robin on all paths. In the second round it is important that the duplication packets are not transmitted on the same path as the first transmission. This ensures a higher loss resilience in case of a path failure. In case of two paths the easiest strategy is to place the entire frame on both paths in parallel.

Packets in Figure 15 that are transmitted redundantly are shown in blue and packets that benefit from the Full Packet Duplication transmission strategy are shown in red. In case of a single packet loss, this transmission strategies significantly improves the Time To Delivery (TTD) and doubles the required throughput on the link as indicated by TO.

In the scenario in Figure 16 path 2 fails after the transmission of packets #2 and #4. The SACK for these packets on this path are also lost. The transmission of all packets succeeds on path 1. The packets #2 and #4 are redundantly transmitted on path 1 but are not acknowledged on path 1 because the CumACK information of previous packets already indicated the successful arrival. The packets #6 and #8 benefit from the full packet duplication mechanism and are acknowledged on path 1 after the successful arrival.

4.4 Deadline Based Preemptive Retransmission

The goal of Deadline Based Preemptive Retransmission (DPR) is to combine the advantages of the Fast Retransmission and Full Packet Duplication transmission strategies to create a trade-off between link utilization and reliability.

DPR allows fast retransmissions until the retransmission deadline D_R. At D_R, packets that are not yet acknowledged or fast-retransmitted are transmitted again. To limit

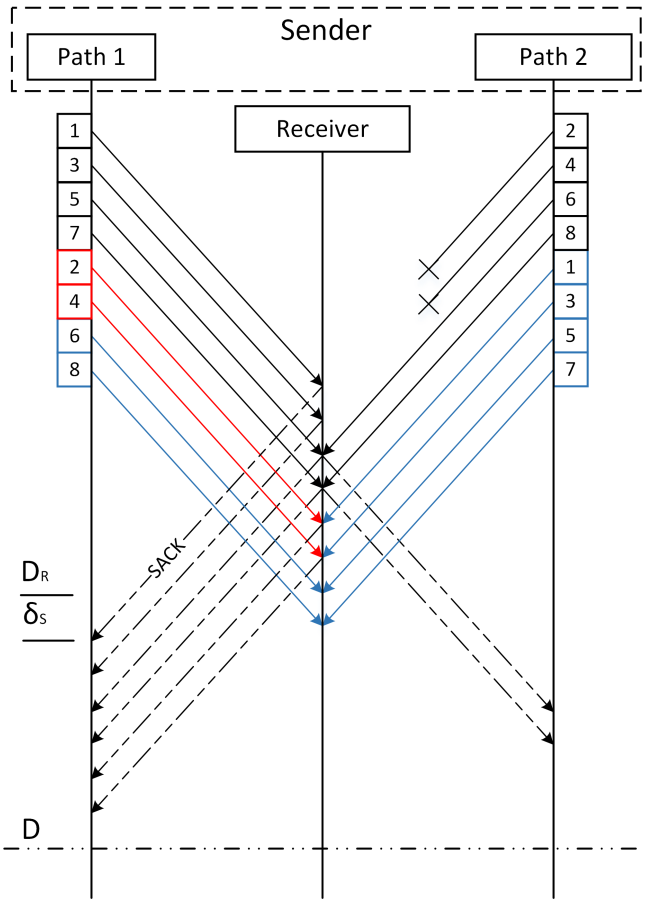

Figure 15: Full Packet Duplication

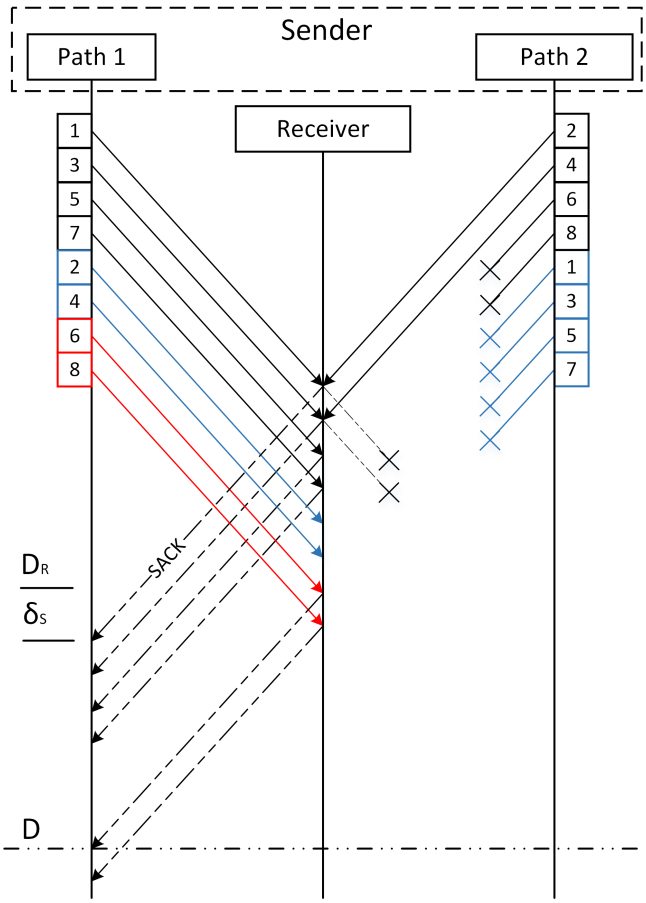

Figure 16: Full Packet Duplication on link failure

the transmits all packets are transmitted at most two times. In Full Packet Duplication all packets are transmitted multiple times in all network conditions and regardless of a successful first transmission. With DPR, the second transmission is postponed so that if the first transmission is successful, an unnecessary retransmission of that packet is avoided. If available DPR uses a path that is distinct from the path of the first transmission to increase resilience against path failures.

Figure 17 shows a network condition that allows packet #2 to be successfully fast-retransmitted within the deadline. The loss of packets #5 and #7 is not detected with a subsequent frame. At D_R packets #5 and #7 have not yet been acknowledged and are therefore transmitted for the second time. Packet #2 has been fast retransmitted already and is therefore skipped. All other packets are already acknowledged at D_R and therefore skipped.

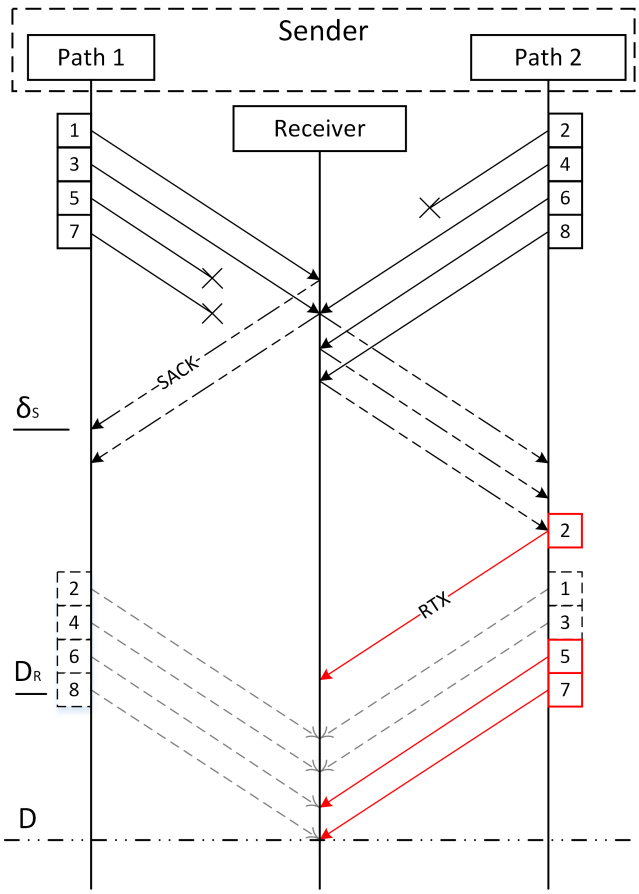

Figure 17: Deadline Based Preemptive Retransmission

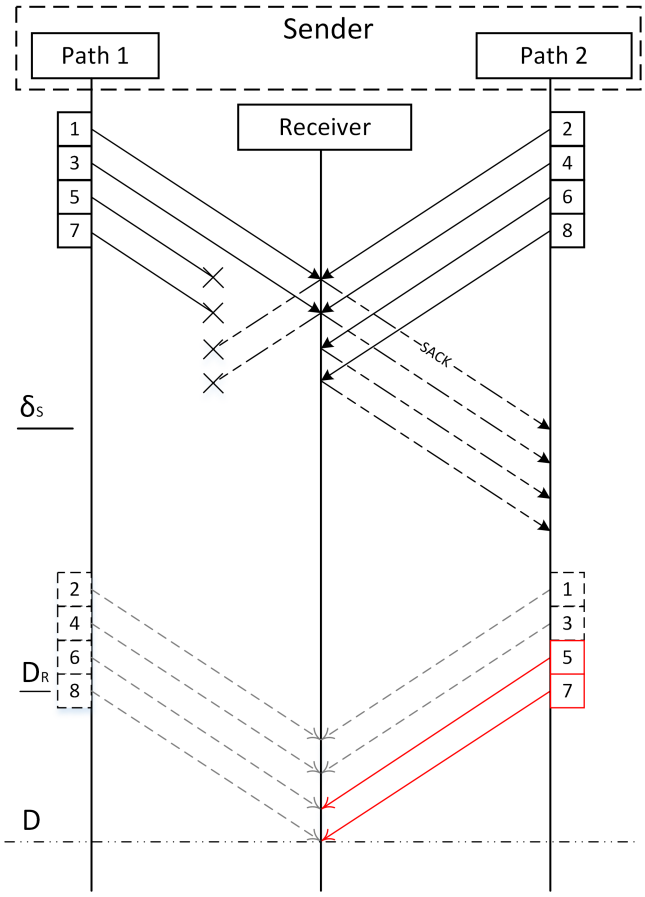

Figure 18: Deadline Based Preemptive Retransmission on link failure

In this scenario Fast Retransmission would lose packets #5 and #7 because their loss would not have been detected in time. With Full Packet Duplication, packets #1-#4, #6 and #8 would be transmitted unnecessarily a second time.

In Figure 18 Path 1 fails after the transmission of packet #3 and all consecutive packets and their SACKs on this path are lost. The lost packets are not detected on this path and according to the rules of CMT they can not be retransmitted on Path 2. At D_R the already acknowledged packets #2, #4, #6 and #8 are skipped. Packets #5 and #7 are retransmitted at this point according to the rules of DPR. Even though the

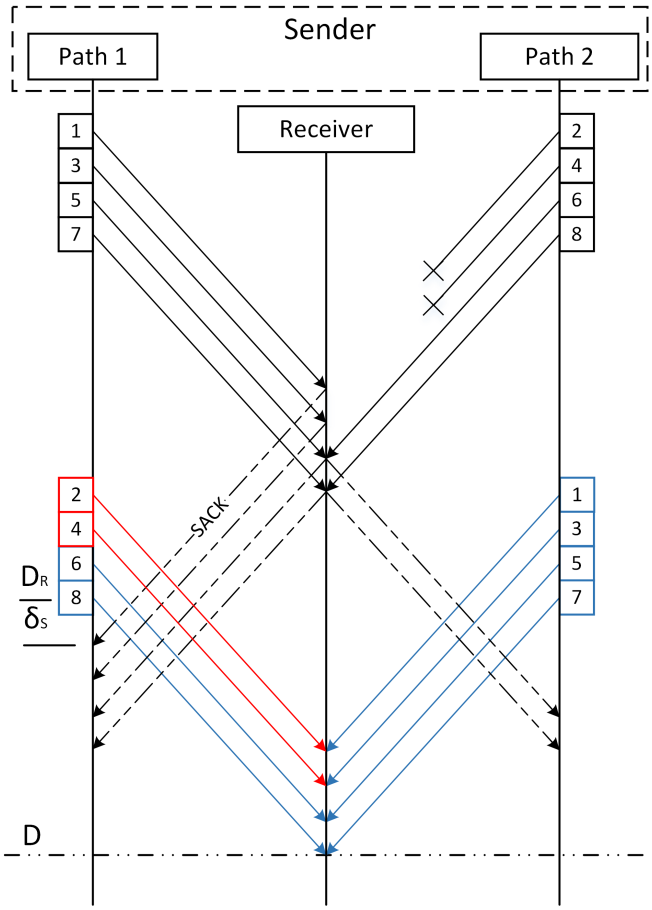

Figure 19: Deadline Based Preemptive Retransmission on slow path

SACKs for #1 and #3 are lost the SACKs on Path 2 contain the information of the successful acknowledgment of these packets. They can therefore also be skipped. [9]

When, as shown in Figure 19, the network is slow, so that the first SACK arrives after the retransmission deadline D_R, no packets are acknowledged before D_R. This means that all packets are transmitted again at D_R. This is similar to Full Packet Duplication with a delay in between. In this scenario Full Packet Duplication is the better strategy because the time to delivery is lower.

An implementation might switch between strategies according to the network conditions. DPR provides good strategy that allows efficient link utilization if the network

[9] This is not valid in general. If packet #1 and #3 arrive after #8 the SACK acknowledging #8 does not contain the acknowledgment of #1 and #3. In this case #1 and #3 are unnecessarily retransmitted at D_R.

is fast while still providing full redundancy if the network is slow. This is also true if the network conditions are changing during the transmission.

4.5 Model Verification

There are various techniques to verify a given model. The model can be proven analytically, simulated, run in an emulated environment or directly implemented in a real-world application.

To analytically investigate the model it is represented in precise mathematical formulae so that the correctness of the model can be proven formally. This requires a precise knowledge of the system and the behavior of the environment.

The behavior of the model can also be simulated in a simulation environment. Various network simulation frameworks exist that support a broad range of network protocols. These protocols and mechanisms can be extended within the simulation framework to investigate their behavior by running experiments. Simulation has the advantage to have control over time within the simulation. The simulation can be stopped at any time and the behavior investigated in detail. Networking models for common networking protocols exist for example in the network simulator ns-3 [89] and OmNET++ [90]. The SCTP model in ns-3 does not contain recent updates like CMT which made it unpractical for this work. The SCTP model in OmNET++ contains many of the recent extensions. [10]

The model may also be implemented in a real-world application and the experiments run in an emulated network environment or directly in a real-world network. An emulated network has the advantage to give full control over the network characteristics allowing the model to be evaluated in different network conditions. The disadvantage is that choosing accurate parameters requires good knowledge of the target environment.

The author chose to adapt a real-world implementation and emulate the test network. The implementation as well as the experimentation setup are described in the following section.

[10] An initial simulation was implemented in OmNET++ in the course of this work but the default SCTP model seemed to behave incorrectly so the author refrained from using the simulation.

5 Implementation

This section presents the implementation of the various packet duplication methods as described in Section 4. The implementation consists of two main components. The multimedia part is implemented in a GStreamer [91] pipeline and the networking part using SCTP is implemented as a GStreamer plugin using the userspace SCTP implementation `usrsctp` [92] which was extended by the DPR duplication strategy.

The major goal of the implementation is to provide a testing environment to evaluate and quantify the characteristics of the duplication strategies introduced in Section 4.

This section describes the multimedia part of the implementation. It describes the basics of the GStreamer framework, the multimedia pipeline used for the test setup and the GStreamer SCTP plugin that was implemented and configured as part of this thesis.

5.1 GStreamer Framework

GStreamer [91] is an open source framework to construct processing pipelines for multimedia data. Such a multimedia pipeline as shown in Figure 20 consists of various *elements*. Each element performs a specific function and passes the data in form of a *buffer* to the next element in the pipeline. Elements are connected via *pads*, where the emitter of multimedia data is called the *source*, also short *src* and the recipient of the data is the called the *sink*. At the start of the pipeline reside one or more source elements and the pipeline terminates at one or more sink elements. Intermediate elements are in general called filter elements.

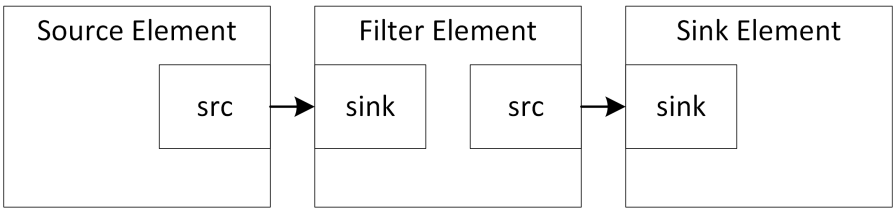

Figure 20: Basic Elements in a GStreamer Pipeline

GStreamer can be used for simple playback of a file, where the source element is a `filesrc` and the sink element for example a playback sink using the audio driver. GStreamer can also be used, for complex pipelines where multiple data flows, like audio and video data exist and may be split at one point and merged at another point of the

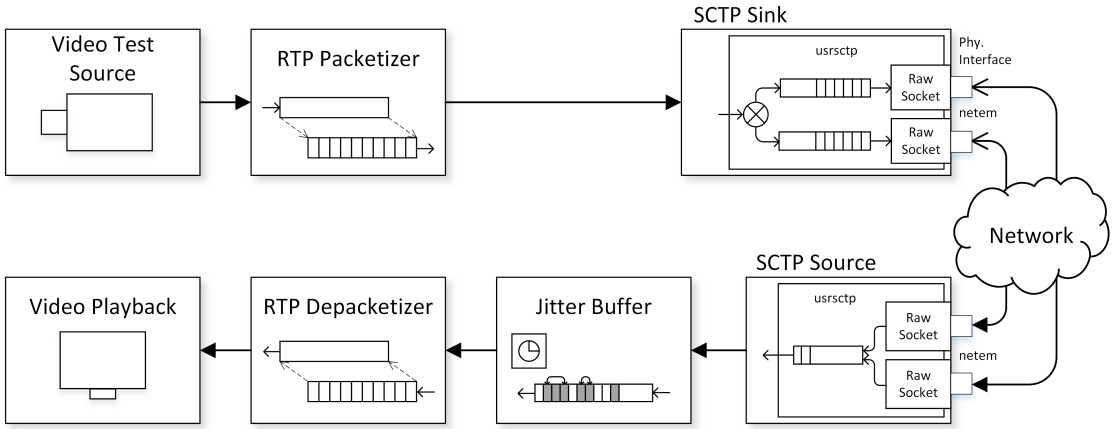

Figure 21: The test setup in GStreamer

pipeline. For multimedia streaming the pipeline terminates — at the sender side — at a network sink like `udpsink`, that sends the data over the network to the receiver, where another pipeline starts with a network source like `udpsrc`.

The framework offers different elements for specific tasks like video and audio codecs, network elements to transfer data over the network, container packager to store audio and video in the same file and many third party vendors offer elements i.e., to control specific hardware.

5.2 Multimedia Pipeline and Test Setup

For the test setup a GStreamer pipeline as schematically shown in Figure 21 is used. At the start of the sender pipeline the video source element `videotestsrc` generates a test pattern. This raw video data is directly passed to the RTP payloader and packetizer element `rtprawpay`. [11] The RTP payloader splits the data into slices of MTU size and attaches the RTP header which contains a sequence number and timestamp. The packaged data is forwarded to the SCTP sink `sctpsink` as described in Section 5.3.

The receiver pipeline starts with the SCTP source element `sctpsrc`, which receives data over the network and passes it on to the jitterbuffer `rtpjitterbuffer`. After a preconfigured time the packets are passed on in equidistant intervals for smooth playback. The `rtprawdepay` element acts as an RTP depacketizer which takes the

[11] In a real-world setup the video stream would be passed through a video encoder before assembling it for RTP transmission.

data fragments and reassembles the video frame which is passed on to a video converter element `videoconvert` which converts the stream into a suitable RAW format and passes it to the video playback element `ximagesink`.

To compare the performance of streaming with SCTP against streaming with UDP, a similar pipeline for UDP was created. The pipelines for UDP contains an `udpsink` instead of the `sctpsink` and an `udpsrc` instead of the `sctpsrc` [12]. The rest of the pipeline is identical.

The application containing the GStreamer pipeline provides many options to enable running all implementation setups from a testing shell script.

To measure the performance of the implementation two standalone computers are used. Both machines run Ubuntu Linux 16.04 with a patched real-time kernel. Both computers are time synchronized with the use of the Global Positioning System (GPS), the Pulse per Second (PPS) API of the Linux Kernel and the Network Time Protocol (NTP) [13]. The computers are connected with two Ethernet links. For development, this setup was run first in two VMware Virtual Machines (VMs), running Manjaro Arch Linux, connected with internal network links. To emulate real-world network conditions the Linux kernel network emulator `netem` [93] is used on the egress connection of both machines. The emulation is done by manipulating the outgoing queue of an interface by performing statistical packet drops, and applying packet delays according to the configuration of the `netem` queue.

5.3 GStreamer SCTP Plugin

In order to test SCTP for multimedia streaming purposes the author implemented a new GStreamer plugin containing the two networking elements `sctpsink` and `sctpsrc`. The plugin has a similar structure to other GStreamer networking plugins like the TCP plugin, although it was not a goal to provide a universally usable SCTP network plugin. In particular functionality that could not be fit into the GStreamer design structure or would have required significant changes, is implemented by hard-coding certain values. Different frame priorities are not considered in this implementation. Still, it is feasible to extend the prototype for general purpose use later on.

The `sctpsink` element terminates the sender pipeline and sends each received buffer over the network to the peer SCTP endpoint. When the GStreamer pipeline is started,

[12] The later described *udpdupl* setup additionally uses a *funnel* element in GStreamer to forward the RTP packets from two *udpsrc* elements to the jitterbuffer

[13] The exact mechanism of this time synchronization is not part of this thesis.

`sctpsink` creates a client socket, sets all socket options according to the configured duplication strategy and connects to the SCTP server. Already at startup multiple path IP addresses are defined. Once the client socket is connected to the server, all incoming buffers are sent to the server via the SCTP socket. When DPR is enabled the retransmission deadline is calculated and passed on to the socket as well. If `sctpsink` receives a list of buffers to send, the I-Flag is set on the last buffer in the list to trigger immediate acknowledgment.

The receiver pipeline starts with the `sctpsrc` element. When the pipeline is started, the `sctpsrc` element opens an SCTP socket as a server and waits for incoming connections. Once the association is established, all incoming chunks are passed on as a GStreamer buffer to the subsequent element in the pipeline.

Many SCTP socket options are available as an option to the GStreamer element to allow its use for testing various duplication strategies.

5.4 SCTP Socket

SCTP is supported by the major operating systems, Linux, Windows, MacOS and the BSD setups. The kernel implementation in Linux [94] does not support many of the recent SCTP extensions that are required to investigate the mechanism discussed earlier in this work. An userspace implementation offers portability and easier development. The author chose the userspace implementation `usrsctp` to implement DPR.

5.4.1 usrsctp

The reference implementation of SCTP was originally implemented in FreeBSD and has been extended and ported to other operating systems. The FreeBSD version of SCTP is still extended and new developments and extensions are commonly developed in there. `usrsctp` [92] uses the SCTP implementation of FreeBSD with an abstraction layer to use in various operation systems. `usrsctp` regularly receives new updates from the FreeBSD code and is also used for the Web Real-Time Communication (WebRTC) data channel in the OpenWebRTC Project [95].

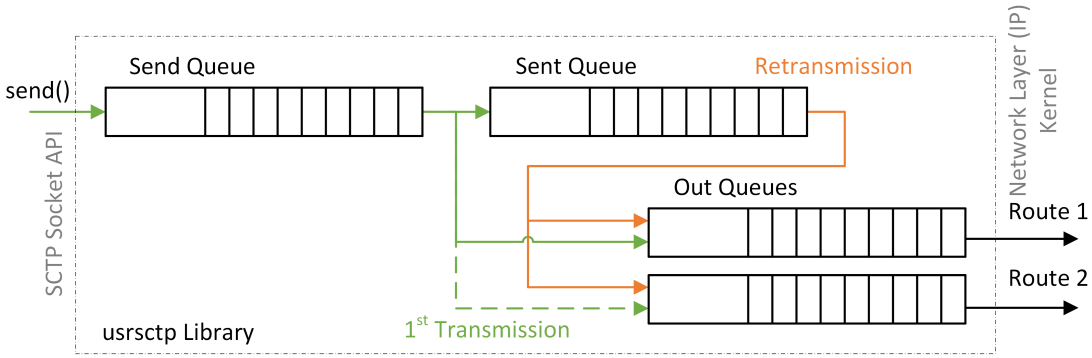

Figure 22: The test setup in GStreamer

5.4.2 DPR in usrsctp

The author forked the usrsctp implementation on GitHub on July 4th 2017 to implement DPR. The author's version of `usrsctp` can be found at [96].

`usrsctp` complies widely to the sockets Application Programming Interface (API) extensions for SCTP [97] and can therefore be used like a regular networking socket.

Figure 22 shows the queuing structure in usrsctp. At the `send()` call, `usrsctp` stores the incoming data in the *send queue*. When a chunk gets scheduled for transmission it moves to the *out queues* of the corresponding network interface and is placed in the *sent queue*. When a retransmission is triggered by a triple duplicate acknowledgment or a RTO the data in the *sent queue* is processed accordingly. If PR is enabled, also the conditions for abandoning chunks are validated and processed at this point.

In order to enable DPR the socket API was extended to pass the retransmission deadline to the socket as a parameter for each chunk. This implementation passes the timestamp for the retransmission deadline D_R in the format equivalent to the result of `gettimeofday()` [14]. `usrsctp` checks if the parameter is set and starts a timer that fires at the retransmission deadline. If the retransmission deadline for multiple chunks are identical or very close together ($< 5ms$), no additional timer is started [15]. In the timer routine the *sent queue* is checked for chunks whose retransmission deadlines have passed and have not yet been retransmitted. Chunks that meet the DPR conditions

[14] `gettimeofday()` is officially deprecated but still used inside `usrsctp`. The author therefore uses the same function to reduce inconsistency in case of a system time adjustment.

[15] Due to the system abstraction layer, timers in `usrsctp` are not Portable Operating System Interface (POSIX) timers but merely a regularly ticking thread that checks if the target time has passed. This lead to inaccuracies in timing for the retransmission deadline timer.

are marked for retransmission which is initiated at the end of the timer routine. [16] This checking for passed timestamps is analogous to the way `usrsctp` checks for passed timestamps when handling partial reliability.

5.4.3 Modified Congestion Control in usrsctp

Loss-based congestion control as applied in SCTP does not allow the CWND to grow properly if the Packet Loss Rate is high on the link. This significantly limits the throughput of an SCTP sender. In the model presented earlier in this work the effects of congestion control throughput limitations are not considered. To appropriately validate the given model the congestion control in `usrsctp` was modified so that the CWND never falls below a limit of $125 \cdot MTU$. This allows the sender in the experimental setup to always send sufficient data and the throughput is never limited. In Section 7.1 the results for both versions, with modified and the regular congestion control are presented for comparison.

[16] The retransmission is in fact called directly from within the timer context which the author considers bad practice. This behavior is common within `usrsctp` though. Lacking a main loop and a messaging system this pattern was as well used in the implementation for DPR.

6 Model Evaluation

With the experiments conducted in this thesis, the author evaluates the characteristics and performance of the duplication strategies introduced in Section 4. The key performance metrics introduced in Section 4.1 are used. Measurements at certain points along the multimedia path are done to obtain the necessary data for the metrics. This section explains the methodology to perform these measurements.

The following implementation setups that represent the duplication strategies are compared in the experiments. The trailing parentheses define the scenario name that will be referred later on by the code and by plots.

- UDP single-homed (udp)
- UDP with packets duplicated to all paths (udpdupl)
- SCTP single-homed with PR (single)
- SCTP multihomed with CMT and Buffer Splitting (cmt)
- SCTP CMT with DPR extension (dpr)

These setups represent the duplication strategies as follows: the reference without any duplication is the *udp* setup, which uses UDP for transmission of the multimedia data. The Fast Retransmission strategy is represented by the *single* and the *cmt* setup. The *single* setup is the default SCTP setup which has multihoming configured where the secondary path is only used as backup. The *cmt* setup uses also Fast Retransmission to duplicate data but uses both paths concurrently. The Full Packet Duplication strategy is represented by the *udpdupl* setup where the UDP packets are transmitted twice with different source and destination addresses through the same UDP socket. The IP layer underneath routes the packets to individual interfaces. The *dpr* setup uses the Deadline Based Preemptive Retransmission duplication strategy implemented in SCTP.

Socket Options

To further define the setups above, Table 4 describes the socket options used in the setups using SCTP. For the setups using UDP no socket options are modified.

Setup	CMT	Buffer Splitting	Partial Reliability	PR Value
`single`	0	0	TTL	D_R
`cmt`	1	1	TTL	D_R
`dpr`	1	1	TTL	D_R

Table 4: SCTP Socket Options

Following options are enabled for all setups. All chunks were flagged with the unordered (U) flag to reduce receiver buffer blocking, as described in Section 3.8.1. To further reduce buffer blocking the Non-Renegable Selective Acknowledgments (NR-SACK) extension is used. As the data in video streaming are periodic bursts, any limitations for maximum bursts are disabled. As described in Section 3.2 Nagle's algorithm tends to limit the performance for multimedia streaming and is therefore disabled in the socket. To allow some packets to be abandoned instead of indefinitely retransmitted as required for multimedia streaming, the transmission uses the PR extension. All setups use the Time to Live (TTL) version of Partial Reliability (PR) where a PR timeout is set for each chunk and if that timeout has passed, the sender abandons that chunk. The timeout is set to $D - \delta_e$. This is equal to the retransmission deadline D_R while ignoring the processing time needed from the timestamp generation in the RTP packetizer and the actual processing of a chunk in the SCTP socket. The SCTP heartbeat interval for inactive paths is set to five seconds.

The experiments show that if the link is too slow many packets are stuck in the sending queue without being sent even once. Whenever a packet gets scheduled for transmission after the PR deadline has passed it is abandoned right away. The *cmt* and *dpr* setups use the CMT extension and allow the socket to utilize both paths concurrently, while using buffer splitting for a condition adapted path selection. For the *single* setup CMT is not used due to predefined path selection.

6.1 Parameters

To evaluate and compare the different implementation setups and respectively the duplication strategies, the setups are tested under different network conditions, which are referred to as the parameters for the experiments. Table 5 summarizes the notations used in the following.

The following parameters are used: The end-to-end delay δ_e and the Packet Loss Rate on both paths and in both directions. In all test runs δ_e and Packet Loss Rate are set

Notation	Description
δ	Delay
δ_e	End-to-end delay
δ_{gst}	Delay added by queuing in GStreamer
δ_R	End-to-end delay including a retransmission
t	Variable for time
D	Deadline
D_R	Retransmission deadline
τ	Point in time
τ_0	Start of measurement in an experiment
τ_E	End of an experiment
τ_{PF}	Time of a path failure
τ_R	Time a packet is measured at the receiver
τ_S	Time a packet is measured at the sender
ρ	Configured packet drop rate
ϕ	Probability
P	Set of packets
p_i	Packet with the index i
V	Transferred volume in bytes
$V(p_i)$, $V(P)$	Size in bytes of a packet or a set of packets
H	Length of a protocol header in bytes

Table 5: Notations

to identical values on both paths and in both directions respectively. The parameters are changed for each autonomous test run. δ_e is set in steps of 20ms from 20ms to 100ms while the deadline D is set to 200ms in all test runs. The link capacity is not limited so only the link delay has a direct effect on the TTD. The Packet Loss Rate was configured with the Simple Gilbert model offered by *netem*. This model has a good and a bad state. The drop probability in the good state was set to 0% and in the bad state to 100%. The transition probability from good to bad ϕ_{gb} was iterated in the different test runs between 0% and 15%. The transition probability from bad to good is set to 75%. This results in the probability to stay in the bad state ϕ_{bb} with 25%. This means, that if a packet was dropped, the consecutive packet will be dropped as well with a probability of 25%. This is to emulate bulk drops. The overall drop probability

Setup	Dupl. Strategy	Protocol	Link Delay	Packet Loss Rate	DDR
`udp`	-	UDP	$< {}^1\!/_3 D$	ρ	$1-\rho$
`single`	Fast Retransmission	SCTP	$< {}^1\!/_3 D$	ρ	$1-\rho^2$
`cmt`	Fast Retransmission	SCTP	$< {}^1\!/_3 D$	ρ	$1-\rho^2$
`dpr`	DPR	SCTP	$< {}^1\!/_3 D$	ρ	$1-\rho^2$
`udpdupl`	Full Packet Duplication	UDP	$< {}^1\!/_3 D$	ρ	$1-\rho^2$
`udp`	-	UDP	$\geq {}^1\!/_3 D$	ρ	$1-\rho$
`single`	Fast Retransmission	SCTP	$\geq {}^1\!/_3 D$	ρ	$1-\rho$
`cmt`	Fast Retransmission	SCTP	$\geq {}^1\!/_3 D$	ρ	$1-\rho$
`dpr`	DPR	SCTP	$\geq {}^1\!/_3 D$	ρ	$1-\rho^2$
`udpdupl`	Full Packet Duplication	UDP	$\geq {}^1\!/_3 D$	ρ	$1-\rho^2$

Table 6: Expected DDR Results

of the link is further denoted as ρ. It can be estimated in the first approximation with $\rho = \phi_{gb} \cdot (1 + \phi_{bb}) = \phi_{gb} \cdot 1.25$.

Expected Results

To compare the performance the key performance metrics Time To Delivery (TTD), Delivery Within Deadline Ratio (DDR), Transmission Overhead (TO) and Path Failure Impact (PFI), are evaluated and expected results are presented in the following.

Network conditions featuring low link delay allow the regular fast retransmissions to happen within the deadline D. When $\delta_e < {}^1\!/_3 D$, the SACK indicating the loss may arrive before D_R and a retransmission may arrive within the deadline. With $\delta_e \geq {}^1\!/_3 D$ fast retransmission is not applicable anymore and Full Packet Duplication offers better performance due to the reduction of the overall TTD of the packets received.

Table 6 summarizes the test parameters and the expected results for DDR. The *udp* setup does not provide any additional reliability functions so the DDR is expected to be reduced by the Packet Loss Rate ρ. The expected DDR for the *udpdupl* is reduced by ρ^2 which is the probability of a packet being dropped on both paths. The setups *single* and *cmt* utilize the Fast Retransmission duplication strategy. With $\delta_e < {}^1\!/_3 D$, DDR is expected to be reduced by ρ^2 indicating the probability of the retransmission being dropped as well.

Description	Symbol	Equation	Length [Byte]
Media Payload	V_{Pl}		1388
Ethernet Header	H_{Eth}		12
IP Header	H_{IP}		20
UDP Header	H_{UDP}		8
RTP Header	H_{RTP}		12
SCTP Common Header	H_{SCTP}		12
Headers when UDP	$\sum H_{UDP}$	$H_{RTP} + H_{UDP} + H_{IP} + H_{Eth}$	42
UDP Data Packet	$V_{UDP,Data}$	$V_{Pl} + \sum H_{UDP}$	1440
Headers when SCTP	$\sum H_{SCTP}$	$H_{SCTP} + H_{UDP} + H_{IP} + H_{Eth}$	46
SCTP DATA Chunk Header	H_{Data}		12
SCTP DATA Packet	V_{Data}	$V_{Pl} + H_{Data} + \sum H_{SCTP}$	1460
SCTP FORWARD-TSN Chunk	$V_{Fwd,Chunk}$		12
SCTP FORWARD-TSN Packet	V_{Fwd}	$V_{Fwd,Chunk} + \sum H_{SCTP}$	68
SCTP NR-SACK Chunk Header	H_{Fwd}		20
SCTP NR-SACK Packet with n gap ack or dupl reports	$V_{SACK,n}$	$H_{Fwd} + \sum H_{SCTP} + 4n$	76 + 4n

Table 7: Packet and Header Length

The expected TO results are determined by the well known length denoted as volume of the headers involved as well as the known volume of the additional packets sent. The number of additional packets sent is estimated by analyzing at the transmission rules in the specification and explained in the following.

Table 7 shows the various header and packet sizes. It describes how the volume for each packet is calculated. SCTP has a common header that contains amongst other information the source and destination port and a chunk type specific header that varies in length. The NR-SACK chunk contains information about duplicated DATA chunks as well as chunks that were received with a gap to the last sequence number that was cumulatively acknowledged. The length of this chunk differs depending on the network condition. For example if the link is lossy, more gap reports are expected to be sent. The set of packets on the link in forward direction (P_L) contains the data packets from UDP and SCTP as well as the SCTP FORWARD-TSN packets. In reverse direction the NR-SACK packet is part of the set P_R.

The expected TO varies depending on the setup and the configured delay δ_e. The `udp` setup is expected the have lowest TO which consists only of the headers added to the payload data.

$$E(TO)_{udp} = \frac{V_{UDP,Data}}{V_{Pl}} - 1 = 0.037 \tag{20}$$

The `udpdupl` setup duplicates all packets which is expected to results in a TO of:

$$E(TO)_{udpdupl} = \frac{2V_{UDP,Data}}{V_{Pl}} - 1 = 1.075 \tag{21}$$

If the configured delay is short compared to the deadline ($\delta_e < {}^1\!/_3 D$) the expected TO is expected to be identical for the `single`, `cmt` and `dpr` setups. The expected TO is constructed with the following assumptions:

- Packets are dropped with a probability of ρ and are retransmitted once upon detection.
- Packets are retransmitted as-well-as abandoned by sending a FORWARD-TSN chunk because the deadline of the packet expires before the NR-SACK for the retransmission arrives at the sender. This is the case for $\delta_e \geq {}^1\!/_5 D$.
- The SCTP Packet containing the DATA chunk has sufficient space available to store the FORWARD-TSN chunk. This means no extra dedicated SCTP packet has to be sent to transfer the FORWARD-TSN chunk.
- Each unique DATA chunk that arrives at the receiver generates a NR-SACK packet that is transferred from the receiver to the sender.
- The sender is configured with one retransmission. $1 - \rho^2$ packets arrive at the receiver.
- For each DATA chunk a single NR-SACK is transmitted because $\rho^2 \ll 1$ and $V_{SACK,6} \ll V_{DATA}$.
- Each NR-SACK contains on average 1 duplicate or gap report.

$$E(TO)^{<1/3D}_{single,cmt,dpr} = \frac{(1+\rho)V_{UDP,Data} + \rho V_{Fwd,Chunk} + V_{SACK,1}}{V_{Pl}} - 1 \\ = 0.110 + 1.061\rho \tag{22}$$

The expected TO at $\delta_e \geq 1/3D$ is identical for the `single` and `cmt` setups. In this scenario the following assumptions are made:

- The sender performs no retransmissions. $1-\rho$ packets arrive at the receiver.
- Because the NR-SACK does not reach the sender before the retransmission deadline the sender abandons all packets.
- All unique DATA chunks trigger the transmission of a single NR-SACK.
- The NR-SACK contains on average 6 duplicate or gap report.

$$E(TO)^{\geq 1/3D}_{single,cmt} = \frac{V_{UDP,Data} + V_{Fwd,Chunk} + (1-\rho)V_{SACK,6}}{V_{Pl}} - 1 \\ = 0.133 - 0.072\rho \tag{23}$$

At $\delta_e \geq 1/3D$ `dpr` sends all packets twice and abandons all packets. All packets are abandoned like above and it is simplified that all unique DATA chunks trigger the transmission of a NR-SACK packet.

$$E(TO)^{\geq 1/3D}_{dpr} = \frac{2V_{UDP,Data} + V_{Fwd,Chunk} + V_{SACK,6}}{V_{Pl}} - 1 = 1.184 \tag{24}$$

6.2 Measurements

This section presents the procedure to measure the key performance metrics TTD, DDR, TO and PFI.

6.2.1 Time To Delivery

Figure 23 schematically shows the measurement points τ_S and τ_R used to measure the TTD for each packet. The RTP packetizer element in the sender pipeline retrieves the current system time with `gettimeofday()` for every packet. It scales the timestamp by the default RTP clock-rate of 90000ns and writes the result into the timestamp field of the RTP header.

For this one-way delay measurement it was necessary to synchronize the system clocks as precisely as possible. For this setup both computers synchronize to a GPS/PPS time source. These RTP timestamps imposes higher measurement accuracy demands than the ones specified in [36].

The result of `gettimeofday()` is typically 128bit [17] whereas the RTP timestamp field has a length of 32bit. To transfer the system timestamp in the RTP header an offset to the system time is captured at the start of each test run [18]. The *sctpsink* element unpacks the RTP timestamp and calculates D_R for each packet, which is passed to the *usrsctp* socket [19].

On the receiver side, when the packet enters the jitter buffer, the sequence number and RTP timestamp are extracted from the RTP header and stored together with the current system timestamp in a file for later analysis. The difference between the timestamps represent the TTD. The DDR is calculating according to Equation (11).

6.2.2 Delivery Within Deadline Ratio

DDR as calculated in Equation (11) represents the ratio of the number of packets with $TTD_i \leq D$ and the total number of packets sent by the sender.

6.2.3 Transmission Overhead

To measure the TO, the volume transferred on the link in both direction is measured. To measure this the system information of transferred bytes on each interface is obtained

[17] It is constructed of two long int values that are 64bit each, on a 64bit CPU architecture.

[18] The available space could have also been increased by increasing the RTP clock-rate to increment for example in units of 1ms or more.

[19] The calculation of D_R could have been done inside *usrsctp*, but as the test run parameters are known in `sctpsink` it was easier to pass the already calculated D_R.

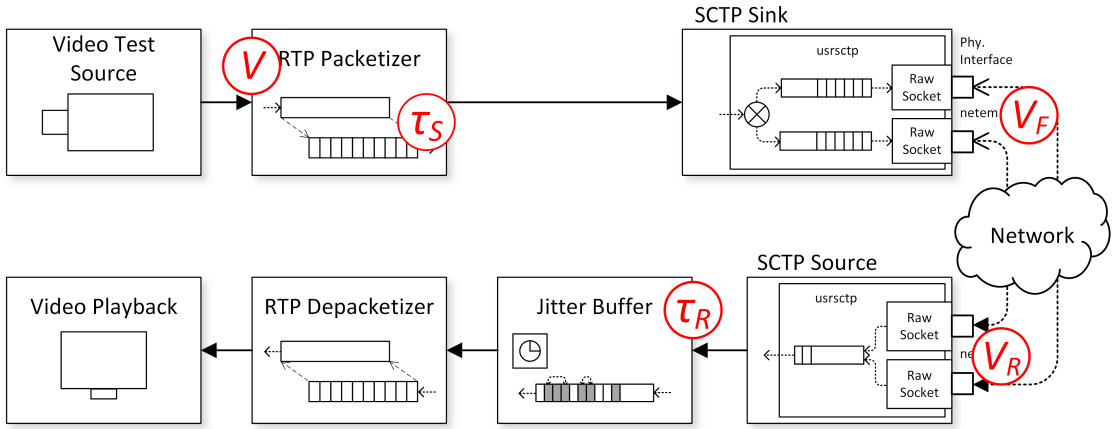

Figure 23: The test setup with the measurement points

before and after each test run and the difference taken as the transferred volume on the link. This information is available in Linux from special files in the `/sys/class/net/` directory. This values also includes other activity on the link. The packet drops in the emulated network are performed in the *netem* queue and the Linux system measures the data after the *netem* queue. The value obtained from Linux represents the volume on the link, without the dropped packets. To calculate the volume, that would have been placed on the link in a real-world scenario $V(P_F)$ and $V(P_R)$, the volume, measured by the system, $V_m(P_F)$ and $V_m(P_R)$ is adjusted with configured packet drop rate. To be more accurate in the measurement the configured drop rate is adjusted more precisely than in the adjustment for ρ, used in the model above.

$$V(P_F) + V(P_R) = \frac{V_m(P_F) + V_m(P_R)}{1 - \rho_{bb} + \rho_{bb}^2 + \rho_{bb}^3} \tag{25}$$

TO is calculated as shown in Equation (15). The volume of the raw data of the stream $V(P)$ is known for the configuration in the test runs. In Figure 23 the measurement points for $V(P_F)$ and $V(P_R)$ are shown as V_F and V_R.

6.2.4 Path Failure Impact

For the measurement of the PFI, the general test setup matches the one described in the previous section. During the test run the primary link is interrupted after a certain time by shutting down the Ethernet interface on the receiver side with the use of Linux `iputils`. To determine the time of the path failure τ_{PF} the sending time of the last

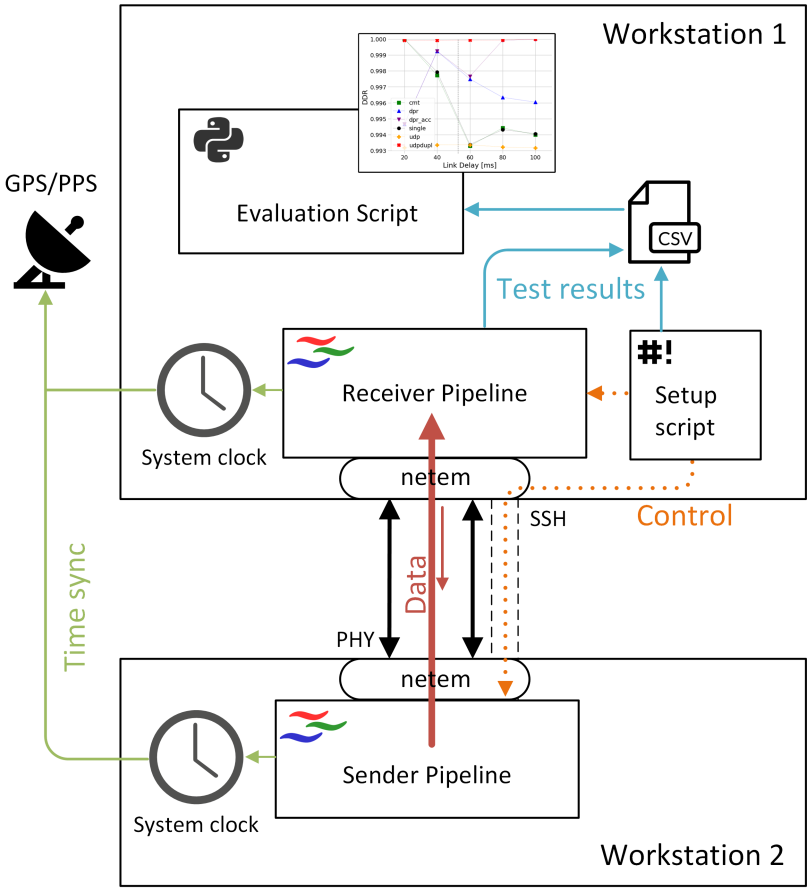

Figure 24: The test execution and evaluation script setup.

successfully transmitted packet needs to be captured. With the previous setup the path a packet p_i was received cannot be determined within the GStreamer pipeline. To determine τ_{PF} the packets on the primary path of the receiver are additionally captured with *Wireshark* and stored for later analysis with the python library `PyShark`. The RTP timestamp of the last packet that arrives at the primary path of the receiver represents the time of the path failure τ_{PF}. The PFI is then calculated according to Equation (13) with a measurement time t_m of 3 seconds.

6.2.5 Test Execution and Evaluation Scripts

Figure 24 illustrates the test execution setup. The test execution scripts are written in `bash`, execute the experiments with a given configuration and collect the measurement data. The test script is executed on the receiving host. The script executes multiple test

runs with different parameters where for each test run the `netem` queue is configured and the GStreamer pipeline is started at the receiver and the sender computer which is accessed via Secure Shell (SSH). During the test execution the SSH connection is idle to minimize the impact on the TO measurements. After the completion of each test run, the measurement data is collected and stored with a record of the parameters of that test run. The measured data is stored in `csv` format for later analysis. This analysis is done by a python script which uses the Python module `pandas` that offers a simplified and extended interface to the `numpy` Python module. The evaluation script collects the data for the individual test runs, calculates statistical values and plots the results accordingly.

7 Results

This section presents the results of the experiments conducted in this thesis: the Time To Delivery (TTD) measured for each packet; the Delivery Within Deadline Ratio (DDR) and Transmission Overhead (TO) measured for a period of interest and the Path Failure Impact (PFI) measured after the occurrence of a controlled path failure.

7.1 Delivery Within Deadline Ratio

This section starts with presenting the parameters used in the experiment and observations in the TTD measurements that are important for the understanding of the results presented afterwards. It continues with the results of the DDR measurements, first presenting the results of the DDR measurements of an SCTP version with modified congestion control and then of the version with the regular SCTP congestion control mechanism. The results are complimented with exemplary plots of the Cumulative Distribution Function (CDF) of the TTD distribution to further illustrate the transmission characteristics.

Parameters

The common parameters for all test runs were configured as follows: All test runs had a deadline configured of $D_c = 200ms$. This deadline configuration was chosen to meet the delay requirements of Section 3.2. The configured deadline only considered the network part of the transmission. When targeting a full system delay of $500ms$ the media processing like capturing and encoding on the sender side and decoding and playback on the receiver side has to be considered. A network delay deadline of $200ms$ leaves enough room for the system to meet the full system delay requirement. To eliminate the effects of the session initialization, only packets sent after $\tau_0 = 3s$ are considered in the evaluation. Table 8 summarizes the parameters of this setup. In each test run, the sender sends 500 frames where each frame was split into 11 packets of $1388B$. With $framerate = 24Hz$ these results in a bitrate of $\approx 2.8Mbit/s$ which represent a common bitrate in real world video streaming [20]. Table 9 summarizes the parameters of the video source.

[20] As of Sept. 2018, YouTube uses approximately this bitrate for 720p video streams. https://support.google.com/youtube/answer/2853702

Symbol	Parameter	Value	Comment
D_c	Configured deadline	$200ms$	
D	Deadline	$158ms$	On the link, without GStreamer
δ_e	End-to-end delay	$[20, 40, 60, 80, 100]ms$	Emulated link delay with `netem`
ρ	Packet Loss Rate	$[0, 0.6, 2.5, 7.2, 8.8, 12.5]\%$	Emulated PLR with `netem`
τ_0	Start of measurement	$3s$	

Table 8: Parameters for DDR measurement setup

Parameter	Value
Number of frames	500
Framerate	$24Hz$
Frame size	$1388B$
Bitrate	$2.8Mbit/s$

Table 9: Parameters of the video source

Offset in the TTD measurements

In the experiment setup the GStreamer pipeline adds a known constant delay to the measurements of TTD. This effect appears because GStreamer synchronizes the pipeline so that the processing starts when data is present at the input of all elements. The RTP timestamp is added to the packet in the RTP packetizer element and passed to the next element. The pipeline gets ready one frame later, adding a constant delay of $\delta_{gst} = {}^{1}/_{framerate} \approx 42ms$. After this delay the packets are handed over to the transport layer. This means that the transport layer receives the packets with an offset of δ_{gst} in all setups regardless of the transport layer protocol used. δ_{gst} represents the time 0 from the perspective of the transport layer protocol and therefore is merely an offset in the measurement. This leads to a deadline for the transport layer of $D = D_c - \delta_{gst} \approx 158ms$. The deadline for the time-based Partial Reliability was configured with the retransmission deadline $D_R = D_c - \delta_{gst} - \delta_e$. SCTP abandons packets after D_R so no packets should arrive at the receiver after D_c. Because this effect is observed in all experiments in all setups the comparison of the experiments is possible. In the DPR setup a similar effect occurs in the receiver pipeline where the packets are kept in the SCTP source element until the RTP jitter buffer element — where the time measurement is done — gets ready for scheduling.

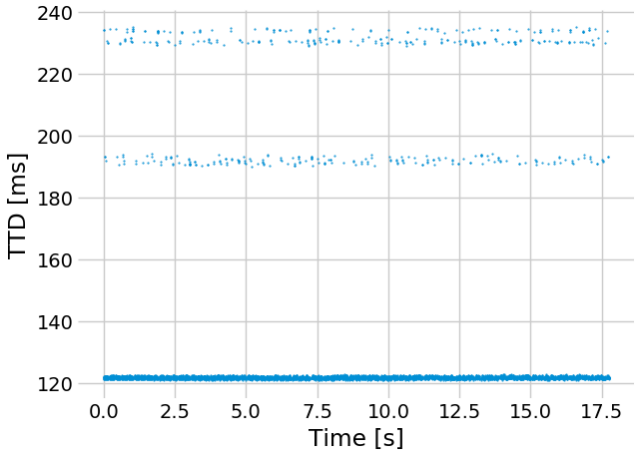

Figure 25: TTD Trace
Parameters: `dpr`, $\rho = 12.5\%$, $\delta_e = 80ms$

Figure 25 illustrates both effects. It presents an exemplary plot of an experimental run with the DPR setup and the TTD measurements for each individual packet over the course of an experimental run. Netem is configured with a delay of $\delta_e = 80ms$ and a drop probability of $\rho = 12.5\%$. As seen in the plot, the first arrival of the packets is measured at $\delta_e + \delta_{gst} \approx 122ms$. In this setup the DPR setup does not trigger any fast retransmissions. Only the DPR retransmissions are triggered. These arrive just before the deadline $D_c = 200ms$. As shown in the plot, various packets arrive after the deadline. These packets have a TTD approximately δ_{gst} larger than the other packets using DPR based retransmissions.

This effect is caused by GStreamer in the experimental setup and not by the implementation of DPR. To accommodate this effect and illustrate the potential performance in the DDR results for `dpr` the virtual setup `dpr_acc` was added which considers a deadline of $D_{acc} = D_c + \delta_{gst} \approx 242ms$.

7.1.1 With Modified Congestion Control

This section presents the results of the DDR measurements for the setups. To better illustrate the transmission behavior, CDF plots of the TTD distribution of different setups at representative parameter sets are presented.

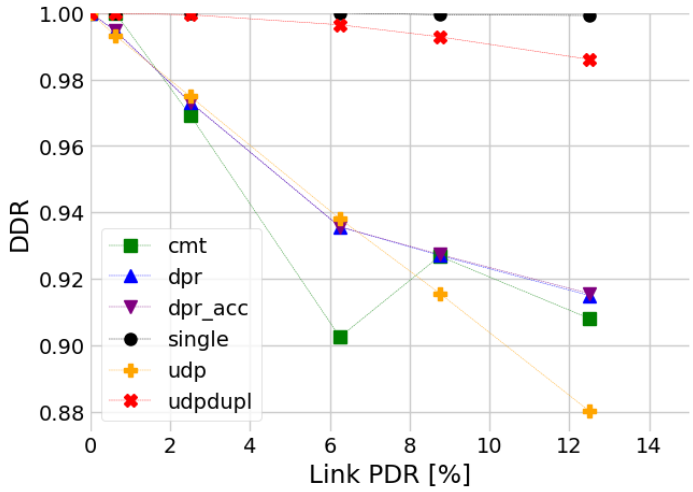

Figure 26: DDR over packet drop rate ρ for different setups
Parameters: $\delta_e = 20ms$

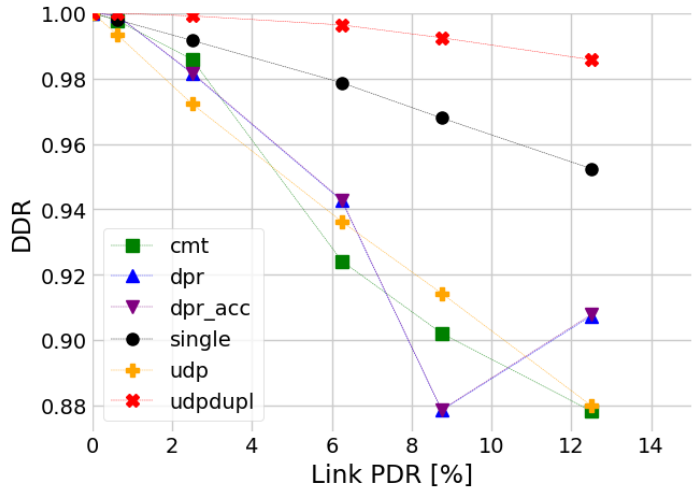

Figure 27: DDR over packet drop rate ρ for different setups
Parameters: $\delta_e = 40ms$

To evaluate the performance of the various duplication mechanisms this section presents the results of the DDR measurements where the congestion control mechanism of the SCTP based setups was modified as explained in Section 5.4.3.

For each parameter set the experiment was conducted 10 times. Each point in the plot represents the overall DDR of all experimental runs with that parameter set. The following plots illustrate the various setups with a single parameter fixed while another parameter is displayed along the x-axis. This allows a direct comparison of the behavior of the different duplication mechanisms and setups. The results are compared to the expected results described in Table 6 in Section 6.2.

Figures 26 till 28 depict the DDR over the emulated Packet Loss Rate ρ for the different setups at fixed delays of $\delta_e = 20ms$, $40ms$ and $80ms$. The UDP-based setups `udp` and `udpdupl` display the expected results. In all test runs the `udp` setup which offers no duplication shows as expected $E(DDR) = 1 - \rho$ and `udpdupl` that uses Full Packet Duplication yields $E(DDR) = 1-\rho^2$. These setups represent the reference to compare the results of the SCTP-based setups against.

At $\delta_e = 20ms$ all setups that offer duplication are expected to reach $E(DDR) = 1-\rho^2$. `single` reaches $E(DDR) = 1$ at all Packet Drop Rates because there is enough time

for more than 1 fast retransmissions before the deadline. [21] At $\delta_e = 40ms$ the DDR of `single` is decreased due to fewer fast retransmissions. At $\delta_e = 80ms$ `single` shows the expected $E(DDR) = 1 - \rho$.

`cmt`, `dpr` and `dpr_acc` result are comparable to `udp` at $\delta_e = 20ms$ and $40ms$ even though $E(DDR) = 1-\rho^2$. At $\delta_e = 80ms$ the result of `cmt` is as expected equal to `udp` while the result of `dpr` is higher and `dpr_acc` yields as expected $E(DDR) = 1 - \rho^2$. This will be investigated further later in this section.

Figures 29 till 31 depict the resulting DDR for all setups over δ_e at $\rho = 0.6\%$, 6.2% and 12.5%. The vertical gray line represents $^1/_3D \approx 53ms$. The `single` setup performs as expected at all Packet Drop Rates (PDRs). It yields at $\delta_e < {}^1/_3D$ a DDR close to $E(DDR) = 1 - \rho^2$ which decreases at $\delta_e \geq {}^1/_3D$ to $E(DDR) = 1 - \rho$. At $\rho = 0.6\%$ `cmt` produces as expected the same result as `single`. At $\rho = 6.2\%$ and $\rho = 12.5\%$ it shows a lower result than expected and at $\delta_e \geq {}^1/_3D$ it yields the expected results of $E(DDR) = 1 - \rho$. `dpr` shows at $\delta_e = 20ms$ at all ρ a lower DDR than expected which increases at $\delta_e \geq {}^1/_3D$ due to the additional retransmission at the retransmission deadline (D_R). `dpr_acc` results according to the expectation $E(DDR) = 1 - \rho^2$ at $\delta_e \geq {}^1/_3D$.

TTD Distribution

The following plots display the CDF of the TTD distribution measured for the packets of a set of test runs. These CDF plots offer a substantiated view of the performance and the transmission characteristics of the analyzed setups. The vertical gray line represents the configured deadline D_c. The value of the CDF when crossing this line indicates the resulting DDR of the set of test runs. A CDF that stops earlier results in a DDR of the last value of the CDF.

Figure 32 illustrates the `single` setup with $\delta_e = 20ms$ for various ρ. The `single` setup has sufficient time to retransmit multiple times before D. `single` reaches in this case a better result than expected of $DDR \approx 1$ at all ρ. Compared to the other setups, as presented in Figure 33, `single` has a better DDR than `cmt` and `dpr` even though multiple fast retransmissions should have been possible. `dpr` is based on `cmt` and has almost identical performance characteristics. Both show fewer initial transmissions at $\delta_e = 20ms$. This occurs if packets are abandoned prior to the first transmission. This indicates that the sender buffer is blocked and some data can not be sent. The reason

[21] PR-SCTP can be configured to abandon packets after a certain time or a certain number of retransmission but not both options at the same time.

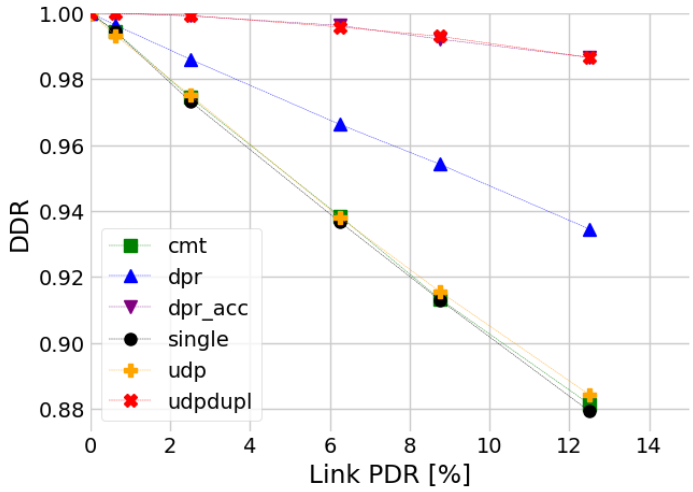

Figure 28: DDR over packet drop rate ρ for different setups
Parameters: $\delta_e = 80ms$

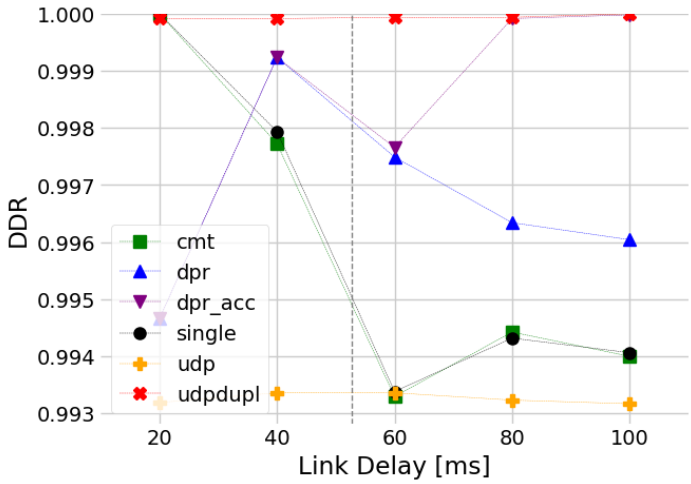

Figure 29: DDR over link delay δ_e for different setups
Parameters: $\rho = 0.6\%$

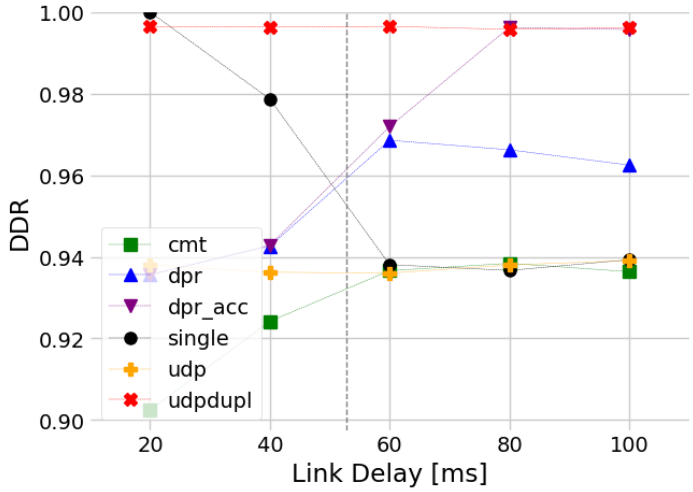

Figure 30: DDR over link delay δ_e for different setups
Parameters: $\rho = 6.2\%$

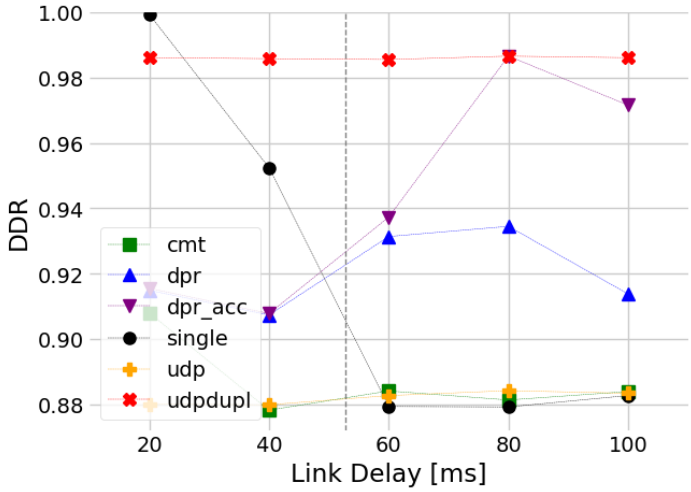

Figure 31: DDR over link delay δ_e for different setups
Parameters: $\rho = 12.5\%$

why fewer packets are sent even though the throughput is not limited by the CWND is attributed to the CMT implementation in `usrsctp` and was not investigated further in this thesis.

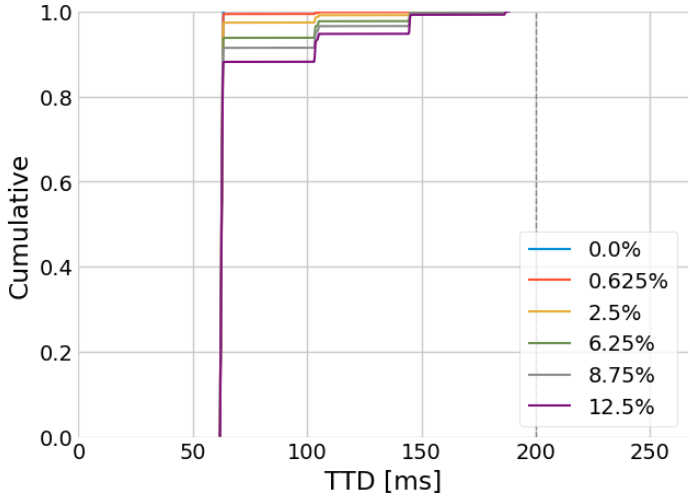

Figure 32: TTD CDF for different packet drop rates ρ
Parameters: `single`, $\delta_e = 20ms$

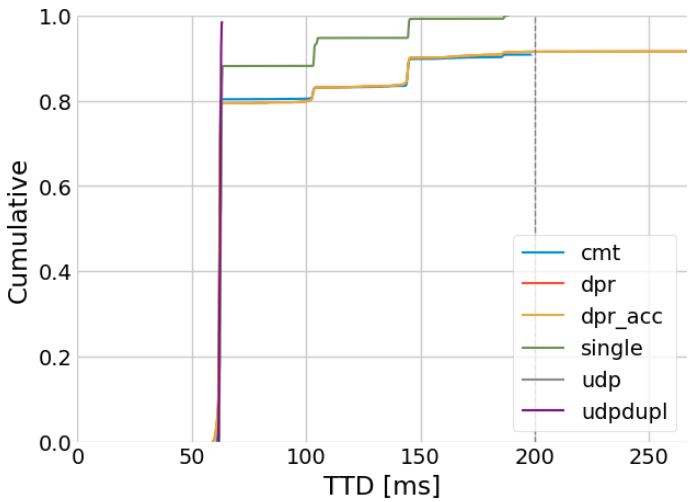

Figure 33: TTD CDF for different setups
Parameters: $\rho = 12.5\%$, $\delta_e = 80ms$

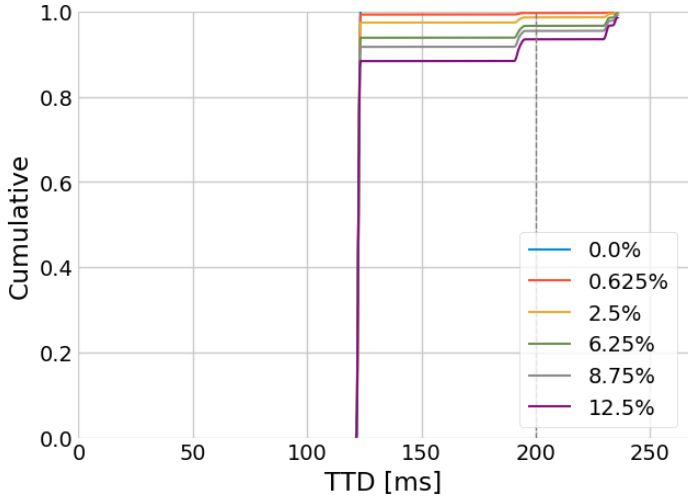

Figure 34: TTD CDF for different DPR
Parameters: `dpr`, $\delta_e = 80ms$

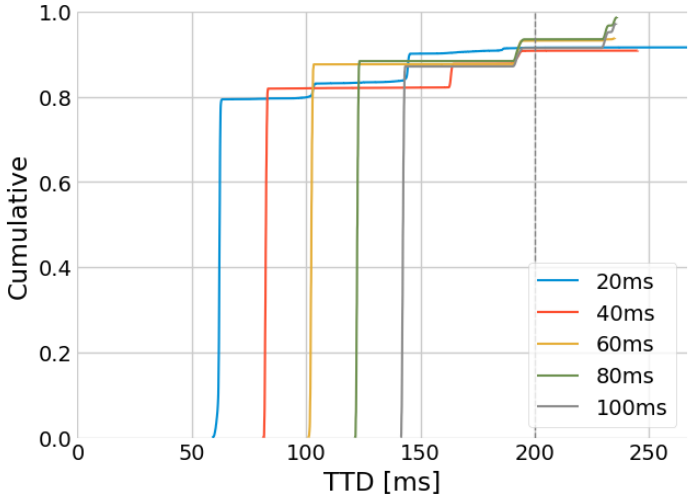

Figure 35: TTD CDF for different link delays δ_e
Parameters: `dpr`, $\rho = 12.5\%$

Figure 34 displays the transmission pattern of dpr at $\delta_e = 80ms$ at various ρ. No fast retransmissions are performed. Just before the configured deadline $D_c = 200ms$ the

additional DPR retransmissions arrive at the receiver. $\delta_{gst} \approx 42ms$ later, the delayed arrivals caused by GStreamer are shown.

Figure 35 depicts the transmission pattern for `dpr` at $\rho = 12.5\%$ at different link delays δ_e. This causes the packets to remain in the `send queue` in `usrsctp`. The current implementation of DPR only checks the `sent queue` which stored packets that have been transmitted at least once. The packets that are not transmitted even once are therefore not considered for a retransmission at the retransmission deadline.

A comprehensive list of test results can be found in the Appendix in Table 11.

7.1.2 With Regular Congestion Control

For the results in this section an SCTP implementation with regular congestion control was used. As mentioned before, loss-based congestion control negatively influences the performance of the SCTP-based setups. Especially with $\delta_e < {}^1\!/_3 D$ and high ρ a throughput limitation is observed due to the sending limitation caused by a small CWND. The following plots show the resulting DDR for the setups and parameters that have been investigated in the previous subsection.

The following plots illustrate the resulting DDR for the investigated setups in the same manner as above.

Figures 36, 37 and 38 display the DDR for all tested setups with variable packet drop rate ρ for a fixed link delay δ_e of $20ms$, $40ms$ and $80ms$. As mentioned above the UDP-based setups show the expected results and are merely used as a reference to compare the SCTP-based setups against.

At $\delta_e = 20ms$ as shown in Figure 36 all setups that offer duplication are expected to reach $E(DDR) = 1 - \rho^2$. The `single` setup performs best of all SCTP-based setups, closely followed by the `cmt` setup. The `dpr` setup yields a DDR of almost 1 at low ρ and reaches a lower result than `udp` at higher ρ.

At $\delta_e = 40ms$ as depicted in Figure 37 again $E(DDR) = 1 - \rho^2$ is expected for all setups that offer duplication. The results indicate that all SCTP-based setups, caused by the congestion control throughput limitation or sender buffer blocking, show a result close to `udp` that offers no duplication. The effects of the limited throughput are visible more clearly later in this section a for example Figure 45.

At $\delta_e = 80ms$ as displayed in Figure 38 those setups that use fast retransmission are expected to yield $E(DDR) = 1 - \rho$ because at $\delta_e \geq {}^1\!/_3 D$ no duplication is provided

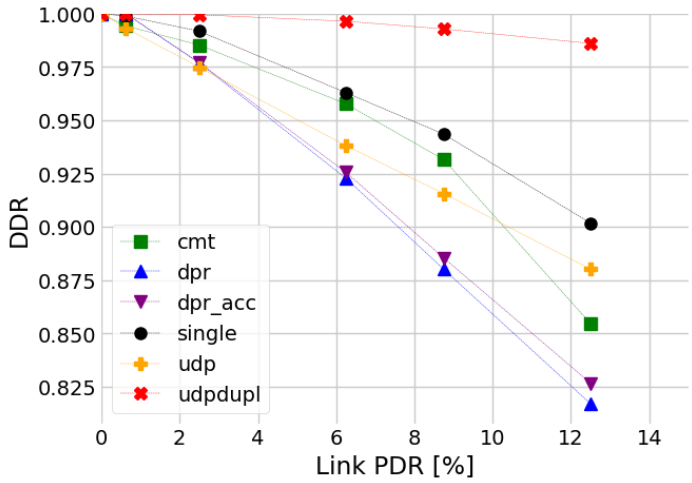

Figure 36: DDR over packet drop rate ρ for different setups
Parameters: $\delta_e = 20ms$

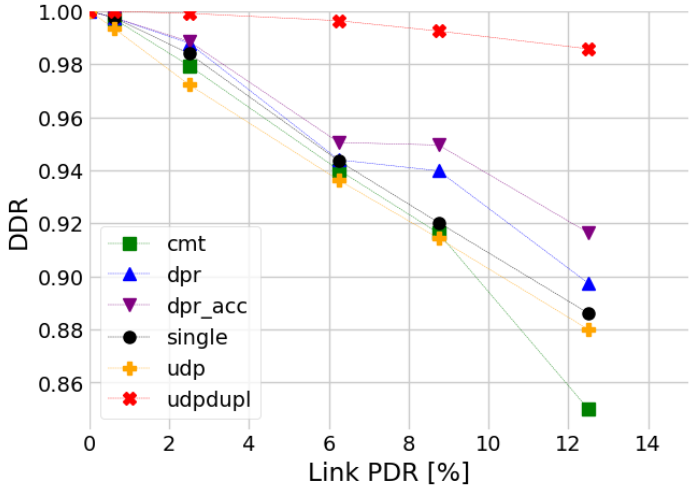

Figure 37: DDR over packet drop rate ρ for different setups
Parameters: $\delta_e = 40ms$

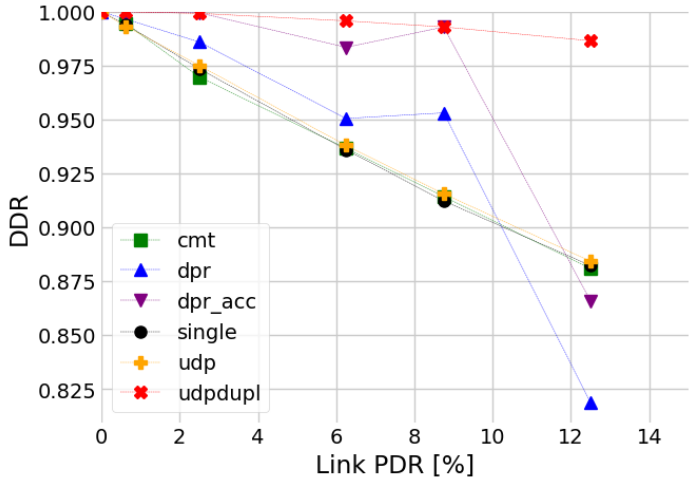

Figure 38: DDR over packet drop rate ρ for different setups
Parameters: $\delta_e = 80ms$

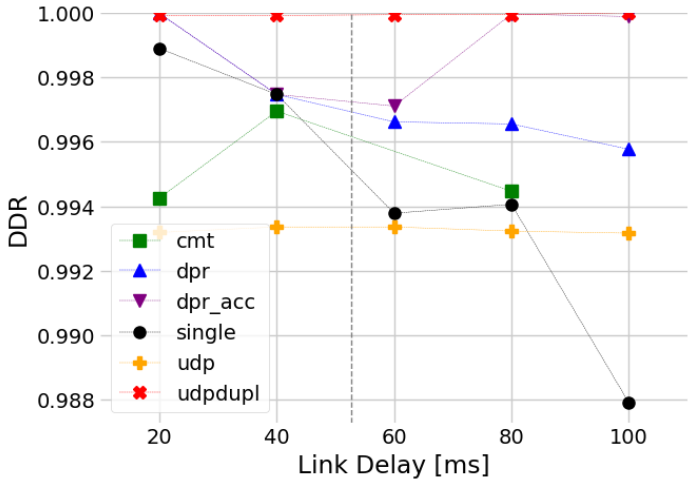

Figure 39: DDR over link delay δ_e for different setups
Parameters: $\rho = 0.6\%$

anymore. The `single` and `cmt` setups that use the fast retransmission mechanism produce this result as seen in Figure 38. The `dpr` setup that uses Full Packet Duplication at this δ_e is expected to yield $E(DDR) = 1 - \rho^2$. It performs better than `udp` and the other SCTP-based setups but does not reach the expected result. When the delay

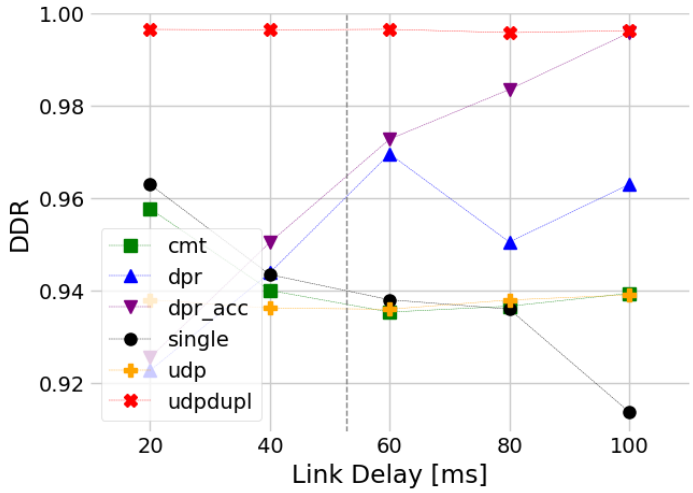

Figure 40: DDR over link delay δ_e for different setups
Parameters: $\rho = 6.2\%$

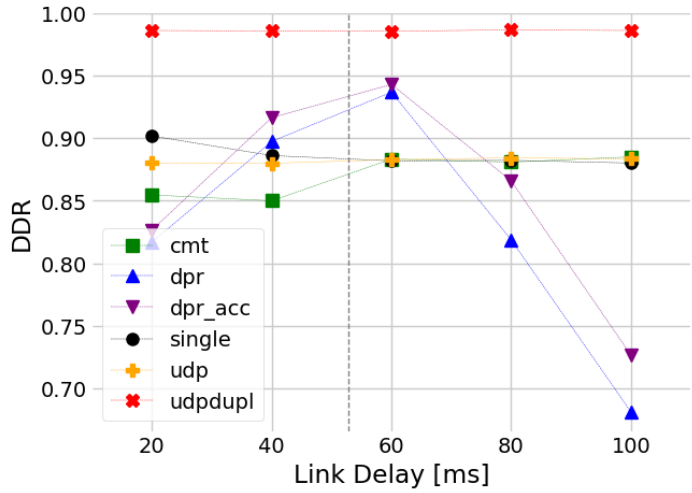

Figure 41: DDR over link delay δ_e for different setups
Parameters: $\rho = 12.5\%$

caused by GStreamer is accommodated as shown in the virtual setup `dpr_acc` the DDR result is close to the expected result but goes down again at high $\rho = 12.5\%$.

Figures 39, 40 and 41 depict the DDR over δ_e at different fixed ρ parameters. The vertical gray line represents $^1/_3 D$. The setups using fast retransmission are expected to produce $DDR = 1 - \rho^2$ at $\delta_e < {}^1/_3 D$ and $DDR = 1 - \rho$ at $\delta_e \geq {}^1/_3 D$. The `dpr` setup is expected to show $DDR = 1 - \rho^2$ at all δ_e.

As illustrated in Figure 39 the throughput limitation caused by the loss-based congestion control at $\rho = 0.6\%$ is small, allowing the `single` setup to perform close to the expected result at almost all δ_e while at $\delta_e = 100ms$ it shows a lower result than expected. The `dpr` setup yields a better result than `udp` at all δ_e and with the GStreamer delay accommodated (`dpr_acc`) a result close to `udpdupl` at higher δ_e.

At $\rho = 6.2\%$ as displayed in Figure 40 the loss-based congestion control has more impact on the DDR result. All SCTP-based setups have a lower result than expected at low δ_e. At $\delta_e \geq {}^1/_3 D$, `cmt` yields the expected result of $E(DDR) = 1 - \rho$ close to `udp` and `single` shows the expected result of $E(DDR) = 1 - \rho$ at $\delta_e = 60ms$ and $80ms$ but a lower result at $100ms$. The `dpr` setup shows as expected a better result than `udp` at $\delta_e \geq {}^1/_3 D$ and `dpr_acc` results close to the expectation of $E(DDR) = 1 - \rho^2$.

At $\rho = 12.5\%$ as presented in Figure 41 the loss-based congestion control exhibits even more impact on throughput. This throughput limitation is clearly visible at $\delta_e < {}^1/_3 D$.

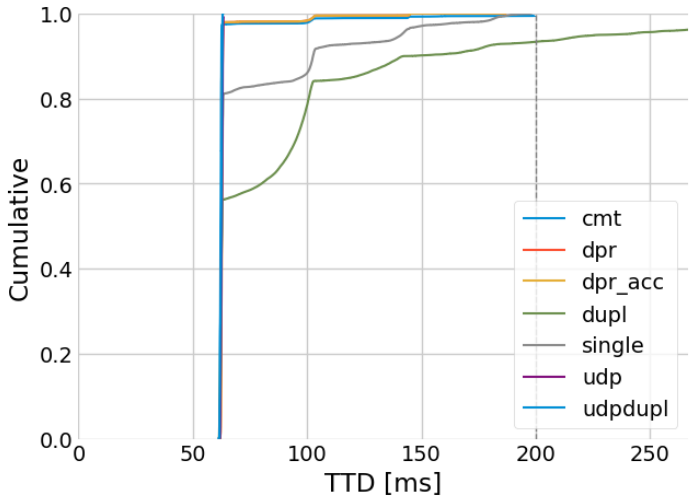

Figure 42: TTD CDF for different setups
Parameters: $\rho = 0.6\%$, $\delta_e = 20ms$

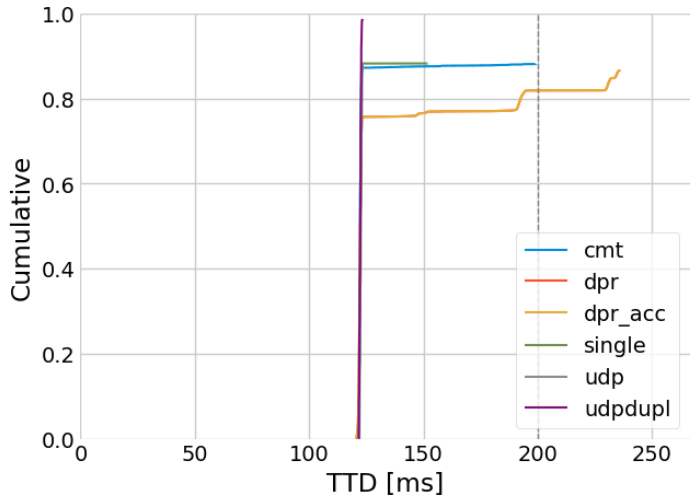

Figure 43: TTD CDF for different setups
Parameters: $\rho = 12.5\%$, $\delta_e = 80ms$

The `cmt` and `single` setups yield at $\delta_e \geq {}^1\!/_3 D$ the expected results of $E(DDR) = 1 - \rho$ similar to `udp`. `dpr` shows better performance than `udp` at $60ms$ but the result goes down at higher δ_e even if the GStreamer delay is accommodated. This is caused as explained by DDR not considering packets that have not been transmitted even once.

TTD Distribution

The following CDF plots of the TTD distribution illustrate in more detail the transmission characteristics of the different setups.

Figure 42 shows the CDF of the TTD measurements for all setups with fixed parameters: $\delta_e = 20ms$ and $\rho = 0.6\%$. Noticeably, for the SCTP-based setups retransmissions arrive at as expected at $102ms$ and $142ms$. This results from the delay added by GStreamer $\delta_{gst} = 42ms$ plus 3 and respectively 5 times the link delay δ_e.

Figure 43 illustrates the CDF of the TTD measurements for all setups with $\delta_e = 80ms$ and $\rho = 12.5\%$, where no fast retransmissions are performed and `dpr` shows the retransmission at D_R. Additionally, it displays the arrival of the DPR retransmission at the retransmission deadline and the delayed arrival caused by GStreamer.

Figures 44 till 47 illustrate the behavior of `dpr` with different fixed parameters.

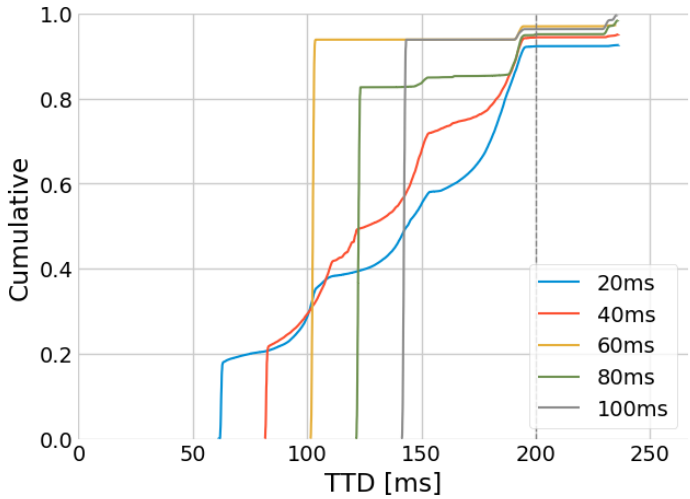

Figure 44: TTD CDF for different link delays δ_e
Parameters: dpr, $\rho = 6.2\%$

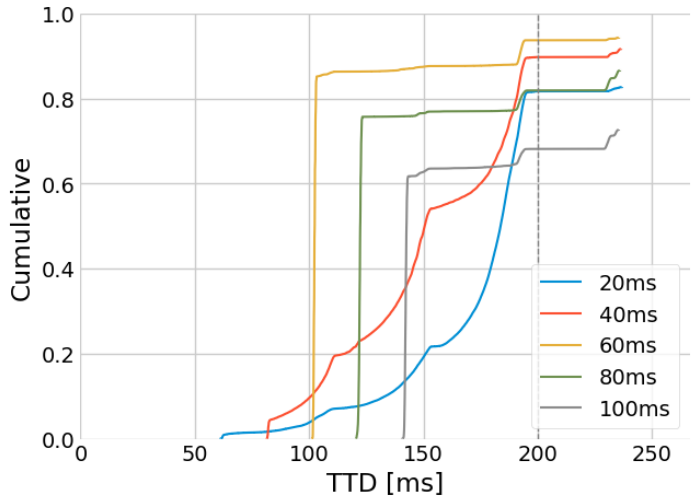

Figure 45: TTD CDF for different link delays δ_e
Parameters: dpr, $\rho = 12.5\%$

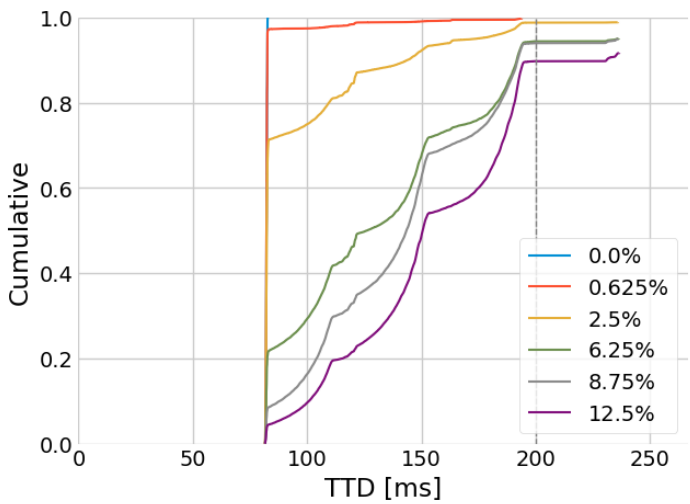

Figure 46: TTD CDF for different packet drop rates ρ
Parameters: dpr, $\delta_e = 40ms$

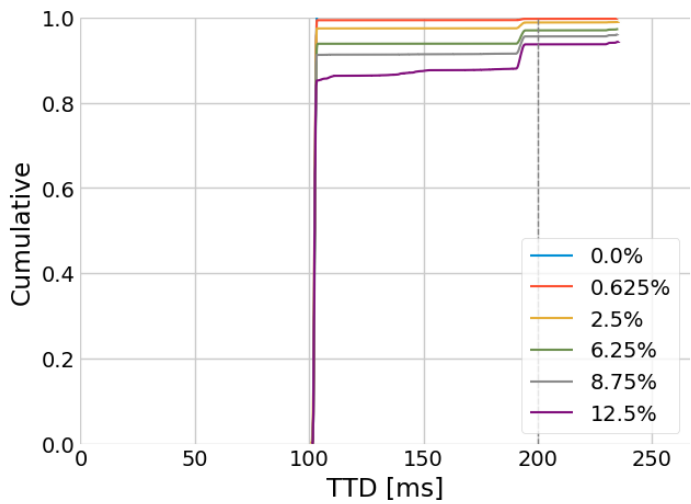

Figure 47: TTD CDF for different packet drop rates ρ
Parameters: dpr, $\delta_e = 60ms$

Figures 44 and 45 display the behavior for different δ_e with $\rho = 6.2\%$ and $\rho = 12.5\%$. In both cases, at lower δ_e many packets are not sent immediately by the sender because at high ρ the loss-based congestion control does not allow the CWND to grow properly

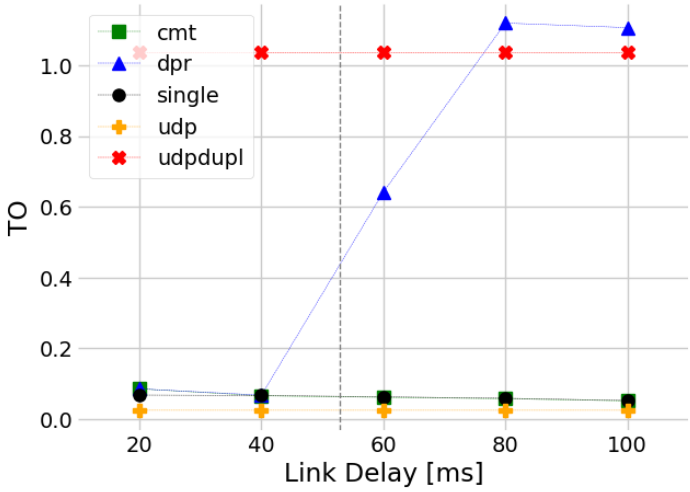

Figure 48: TO over link delay δ_e for different setups
Parameters: $\rho = 0\%$

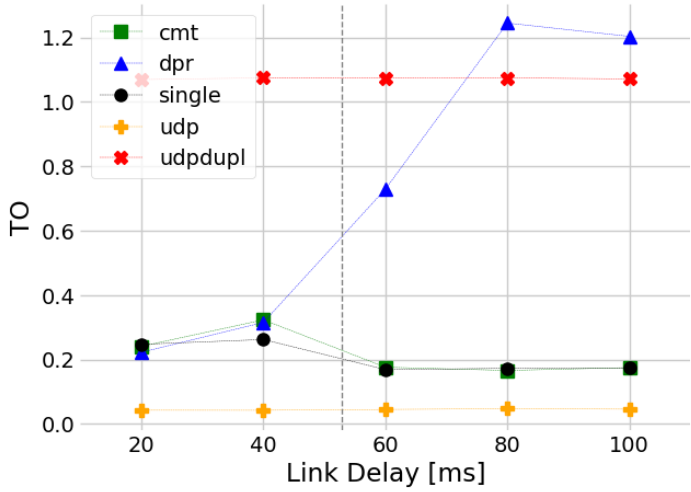

Figure 49: TO over link delay δ_e for different setups
Parameters: $\rho = 12.5\%$

causing a throughput limitation. At $\delta_e \geq {}^1\!/_3 D$ packets are abandoned by the sender before a loss is detected by the sender preventing the CWND from degradation.

Figures 46 and 47 show the behavior for different ρ with $\delta_e = 40ms$ and $\delta_e = 60ms$. At $\delta_e = 40ms$, higher ρ causes a significant throughput limitation whereas at $\delta_e = 60ms$ the losses do not affect the CWND allowing a higher throughput. In both cases many packets arrive right before D_c due to the DPR retransmission.

Table 12 in the Appendix compiles a comprehensive list of test results with regular congestion control.

7.2 Transmission Overhead

The Transmission Overhead (TO) represents the additional transmitted data volume compared to the raw media data generated by the media source. A value of 0 indicates no overhead whereas a value of 1 indicates that the overhead is as much as the media data itself. The TO measurements are only presented for the SCTP implementation with modified congestion control.

Figures 48 and 49 display the resulting TO for all setups at different δ_e with $\rho = 0\%$ and 12.5%. The expected TO is presented in Equations 20 till 24.

Symbol	Parameter	Value	Comment
D_c	Configured deadline	$200ms$	
D	Deadline	$158ms$	On the link, without GStreamer
δ_e	End-to-end delay	$[20, 40, 60, 80, 100]ms$	Emulated link delay with `netem`
ρ	Packet Loss Rate	0%	Emulated PLR with `netem`
τ_{PF}	Time of path failure		
τ_0	Start of measurement	τ_{PF}	
τ_E	End of measurement	$\tau_{PF} + 3s$	

Table 10: Parameters for PFI measurement setup

As expected, the `udp` setup produces a small $TO \approx 0$ and `udpdupl` yields $TO \approx 1$ at all ρ. The SCTP-based setups, `single` and `cmt` show as expected the protocol overhead and retransmissions. Depending on ρ the value of TO is higher at $\delta_e < {}^1\!/_3 D$ where fast retransmission is possible and decreases at $\delta_e \geq {}^1\!/_3 D$. As expected the `dpr` setup shows at all loss rates a low TO at low δ_e which increases at higher δ_e.

Table 12 in the Appendix contains all measured and expected TO results with actual values for the different scenarios and ρ parameters.

7.3 Path Failure Impact

This sections covers the results of the Path Failure Impact (PFI) experiments. During these experiments the primary path is interrupted at τ_{PF} and the TTD measurements in the time period $[\tau_{PF}, \tau_{PF} + 3s]$ are analyzed. The calculated DDR of this period represents the resulting PFI value as presented in Equation (13). For all test runs netem was configured with a loss rate of $\rho = 0\%$ to isolate the effects of the path failure from losses. Table 10 summarizes the parameters of this setup. In the following several exemplary TTD plots are presented followed by a plot that compares the resulting PFI of all setups.

Figure 50 illustrates the resulting TTD for the `cmt` setup. The analyzed period is indicated by the vertical gray lines. The first gray line displays the time of the path τ_{PF} and the second gray line shows $\tau_{PF} + 3s$.

The plot depicts that for `cmt` after τ_{PF} packets stop arriving at the receiver. The situation recovers after some time when `cmt` redirects the data to the secondary path.

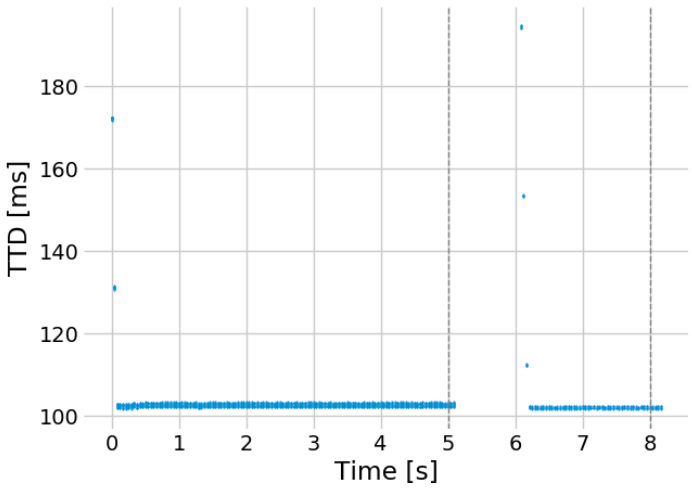

Figure 50: TTD Trace with path failure
Parameters: `cmt`, $\delta_e = 60ms$

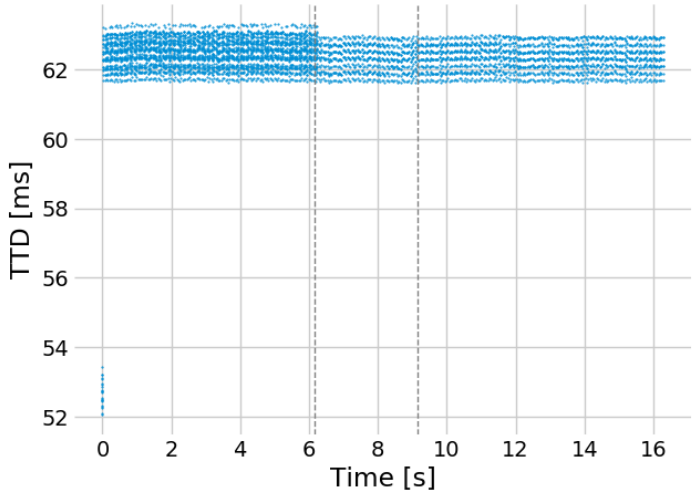

Figure 51: TTD Trace with path failure
Parameters: `udpdupl`, $\delta_e = 20ms$

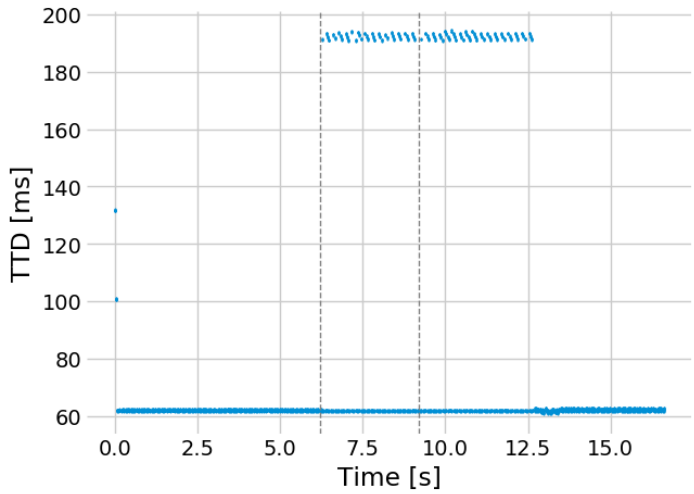

Figure 52: TTD Trace with path failure
Parameters: `dpr`, $\delta_e = 20ms$

For `udpdupl`, as displayed in Figure 51 the transmission continuous as expected uninterruptedly after τ_{PF} while the modified arrival pattern is noticeable. As reflected by Figure 52, `dpr` starts after τ_{PF} as expected to retransmit packets at the retransmission deadline D_R. Figure 53 depicts that `dpr` is affected at higher δ_e by the delay induced by GStreamer.

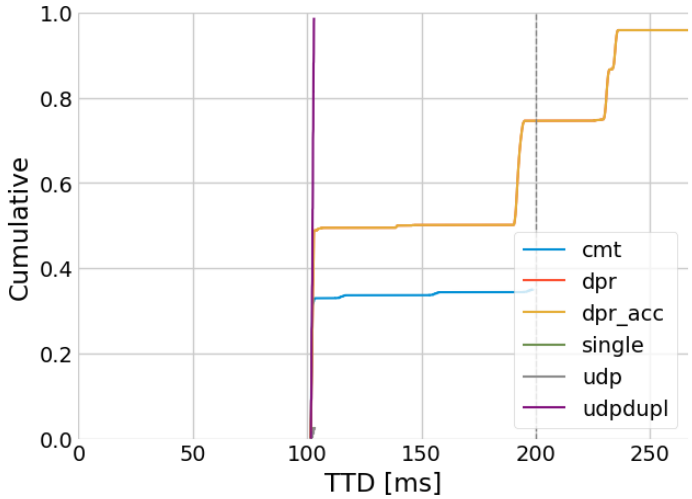

Figure 53: TTD CDF for different setups
Parameters: $\delta_e = 60ms$

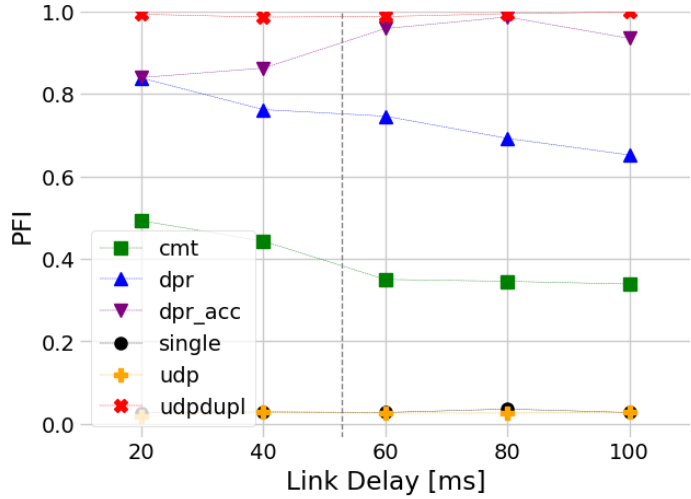

Figure 54: PFI over link delay δ_e for different setups

Figure 54 illustrates the measured PFI for all setups. As expected, `udp` does not provide any path failure protection indicated by $PFI \approx 0$ and `udpdupl` shows almost no impact indicated by $PFI \approx 1$. `single` does not detect the failed path fast enough as described by $PFI \approx 0$. `cmt` recovers quickly after τ_{PF} and yields a positive PFI value but significantly lower than `udpdupl`. `dpr` shows a reduced PFI at $\delta_e \leq {}^1\!/_3 D$ caused by the already explained throughput limitation in `usrsctp`. When the delay caused by GStreamer is accommodated in the virtual setup `dpr_acc` yields at $\delta_e > {}^1\!/_3 D$ a $PFI \approx 1$. Table 13 in the appendix shows the results of all PFI measurements.

It is noticeable that `single` and `udp` do not display $PFI = 0$ even though these setups do not offer any protection against path failure. It turns out that disabling the Ethernet interface in Linux with `iputils` notifies and stops the Wireshark capturing process. Some packets are forwarded by the socket to the GStreamer pipeline afterwards. Because τ_{PF} is captured with Wireshark this value is too early. The actual result for `single` and `udp` is $PRI \approx 0.026$ indicating a small inaccuracy of the τ_{PF} measurement of 2.6%. The measurement results expose that after τ_{PF} from Wireshark between 1 and 3 frames are still received at the receiver.

8 Conclusion

The aim of this thesis was to increase reliability through redundancy in safety-critical real-time video streaming. For this increase in reliability spacial redundancy as provided by multihoming was the main focus of investigation in this work.

Multihoming can increase the link capacity by sharing the load between multiple paths and provide resilience against path failures and packet losses. Intra-flow multihoming is particularly interesting to increase the reliability of a single flow because it allows a single flow to concurrently utilize multiple paths. This thesis analyzed the feasibility of intra-flow multihoming on various layers of the Internet architecture model. Protocols on transport or application layer have been found to allow the necessary functionality. Protocols and mechanisms that implement multihoming like SCTP and Multipath TCP were analyzed for their support the following requirements:

- Intra-flow multihoming
- Performance with latency constrained payloads
- Packet loss recovery
- Path loss recovery
- Mobility support

The transport layer protocol SCTP has been identified as a suitable candidate for latency constrained real-time multimedia streaming with multihomed mobile endpoints. SCTP offers the following functionalities which are crucial in this scenario:

- Multihoming that allows the connectivity of SCTP endpoints through multiple paths that can sustain the failure of paths by using a backup path.
- Multihoming in SCTP also allows mobility by dynamically assigning new paths when due to endpoint mobility some of the existing paths become (temporarily) unavailable.
- Unordered delivery for a subset of packets allows the receiver to pass the data to the application directly upon arrival allowing faster delivery to the application and allows the receiver buffer to be cleared.

- The Partial Reliability (PR) extension allows already sent data to be abandoned by the sender when transmission of the data is impractical because it has become irrelevant to the receiver. Abandoning allows the sender to focus on relevant data and clearing out the sender buffer.
- The Concurrent Multipath Transmission (CMT) extension adds intra-flow multihoming which allows the concurrent utilization of multiple paths in the association.

Further analysis highlights the additional challenges with intra-flow multihoming for packet scheduling and buffer management in the endpoints. The throughput of a subflow might be limited by asymmetric path characteristics. A set of countermeasures is available to improve the overall throughput of the transmission.

The reliability of the stream can be increased by providing redundancy through packet duplication. This thesis models the following three duplication strategies and verifies their performance and characteristics in the experimental setup.

- Fast Retransmission (FRTX) which duplicates on demand and a feedback from the receiver.
- Full Packet Duplication (FPD) which duplicates the entire stream requiring no feedback from the receiver.
- Deadline Based Preemptive Retransmission (DPR) which combines the two previous strategies by retransmitting — not yet acknowledged — packets just-in-time to meet the configured deadline. This strategy was newly developed in the course of this thesis.

To evaluate and compare the performance of these duplication strategies this thesis developed the following key performance metrics:

- Time To Delivery (TTD) defines the one-way delay for the successful delivery of a single packet including the delay for retransmission.
- Delivery Within Deadline Ratio (DDR) describes a metric of one-way connectivity for hard real-time multimedia streaming as a ratio of packets, delivered to the receiver within a given deadline.

- Path Failure Impact (PFI) characterizes the immediate impact of a path failure on the one-way connectivity.
- Transmission Overhead (TO) represents the access data that is sent on the links in both directions compared to net data payload. It includes protocl headers, control messages and duplicated packets.

In order to verify the duplication strategies extensions to various existing software have been made and a verification streaming application was implemented by using the real-world streaming framework GStreamer. GStreamer was chosen due to its modular design which allows the creation of custom streaming pipelines and simple extensibility for various purposes and features through plugins.

A new SCTP network plugin was implemented to enable SCTP transport in a GStreamer pipeline. This plugin provides a sender element (`sctpsink`) which terminates the sender pipeline and a receiver element (`sctpsrc`) which is the initial element in the receiver pipeline. These elements use the user-space SCTP implementation `usrsctp` which was extended to optionally perform the DPR duplication strategy. In the course of the implementation a non-standard compliant behavior of `usrsctp` with severe impact on the performance in the given scenario was identified and fixed.

The investigated duplication strategies were implemented as a GStreamer pipeline which transfers a test video stream over an emulated network between two Linux computers. The required one-way latency measurements are performed by transferring the GPS/PPS time-synchronized system time as part of the RTP protocol header which required modification of standard GStreamer behavior.

The experiments conducted in this thesis yield the following results. If the path delay is short compared to the deadline D, retransmissions triggered by a detected loss are acceptable. Detecting a packet loss at network layer requires at least $2 \cdot RTT$. If the path delay exceeds $^1/_3 D$, retransmitting upon a detected loss does not provide any additional reliability for time-critical communications with hard deadlines. Full Packet Duplication onto multiple paths offers the best reliability even in case of path failure but also adds a significant overhead to the transmission. The newly developed Deadline Based Preemptive Retransmission was shown to provide an efficient trade off between the network throughput of Fast Retransmissions and the reliability of Full Packet Duplication. The experiments also showed that in high-loss networks — like WiFi — loss-based congestion control offered by SCTP is not practical for real-time multimedia streaming as the throughput is significantly limited and in turn the latency increased as shown in Section 7.

8.1 Future Work

The results identify the benefits of packet duplication to increase reliability. These results can be used as a basis for further development which are briefly described in the following.

Loss-based congestion control was shown in this thesis not to be suitable for real-time multimedia streaming in a lossy network like WiFi. The model in this thesis does not consider the influences of congestion control, neither the negative impact on latency caused by the throughput limitation, nor the negative effect on the entire network when a stream is sent out without any throughput limitation. Future work may investigate delay-based congestion control like TCP's Vegas and Bottleneck Bandwidth and Round-trip propagation time (BBR) in conjunction with partial reliability and compare it with the performance of loss-based congestion control. As an alternative to congestion control, the throughput may be adapted to the network condition by controlling the rate of the data in the stream. In multimedia streaming the stream quality can be optimized according to the available link capacity, for instance by accurate monitoring of packet or frame losses. This information is typically transferred from the receiver to the sender with the use of the RTCP protocol. If this contains information about the quality of the individual paths in a multihomed setup, the sender may adapt the stream rate and path selection strategy accordingly. Typically, multimedia streaming uses RTP on top of UDP. Performing this rate control and path selection at application layer makes an implementation and adoptions easier than changing a transport layer protocol that is typically part of the operating system kernel.

Novel protocols like Quick UDP Internet Connection (QUIC) integrate various features from SCTP and MP-TCP at application layer by using UDP as a transport layer protocol. This offers a promising alternative protocol choice for further development. Standardization of QUIC is currently work in progress which can be found in [60]. A multipath extension has been proposed in [98] and a PR extension in [99]. Additionally QUIC offers BBR congestion control [22] and FEC which both may benefit the latency performance in lossy environments like WiFi. Considerations of unreliable streams in QUIC can be found in [100], specifying that the application may decide whether a packet shall be transmitted or not, giving the application full control of the retransmission and duplication behavior.

To more efficiently utilize the link capacity the inherent priorities of the different frame types in a video stream may be used for prioritization of the packets on the network.

[22] As part of the reference implementation in Chromium

This prioritization may be done by applying packet duplication only for reference frames while leaving differential frames unprotected.

Finally, updated key performance metrics may include a transport efficiency factor that combines DDR and TO to show efficiency and the impact of additional traffic on DDR. Furthermore, in addition to the PFI that indicates the performance for a given time after the path failure, a metric may be introduced that indicates the protocol's reaction time to path failures. For example the time between path failure and full service recovery may be used to forecast potential outages and react accordingly.

9 References

[1] J. Wannstrom, "LTE-Advanced." http://www.3gpp.org/technologies/keywords-acronyms/97-lte-advanced.

[2] J. Wannstrom, "HSPA." http://www.3gpp.org/technologies/keywords-acronyms/99-hspa.

[3] "IEEE 802.11ac-2013 - IEEE Standard for Information technology–Telecommunications and information exchange between systems?Local and metropolitan area networks–Specific requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications–Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.." https://standards.ieee.org/standard/802_11ac-2013.html.

[4] T. Dreibholz, "Evaluation and Optimisation of Multi-Path Transport using the Stream Control Transmission Protocol." http://duepublico.uni-duisburg-essen.de/servlets/DocumentServlet?id=27444, Mar. 2012.

[5] K. Yedugundla, S. Ferlin, T. Dreibholz, O. Alay, N. Kuhn, P. Hurtig, and A. Brunstrom, "Is multi-path transport suitable for latency sensitive traffic?," *Computer Networks*, vol. 105, pp. 1–21, Aug. 2016. http://linkinghub.elsevier.com/retrieve/pii/S1389128616301396.

[6] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *2014 IEEE International Conference on Communications (ICC)*, pp. 1222–1227, June 2014.

[7] J. Wang, J. Liao, and T. Li, "OSIA: Out-of-order Scheduling for In-order Arriving in concurrent multi-path transfer," *Journal of Network and Computer Applications*, vol. 35, pp. 633–643, Mar. 2012. http://www.sciencedirect.com/science/article/pii/S1084804511001822.

[8] H. Sanson, A. Neira, L. Loyola, and M. Matsumoto, "PR-SCTP for real time H.264/AVC video streaming," in *2010 The 12th International Conference on Advanced Communication Technology (ICACT)*, vol. 1, pp. 59–63, Feb. 2010.

[9] C.-M. Huang and M.-S. Lin, "The unreliable-concurrent multipath transfer (U-CMT) protocol for multihomed networks," *Telecommunication Systems*, vol. 52, pp. 245–259, Aug. 2011. http://link.springer.com/article/10.1007/s11235-011-9556-7.

[10] J. Wu, C. Yuen, N. M. Cheung, and J. Chen, "Delay-Constrained High Definition Video Transmission in Heterogeneous Wireless Networks with Multi-Homed Terminals," *IEEE Transactions on Mobile Computing*, vol. 15, pp. 641–655, Mar. 2016.

[11] E. Blanchard, *Introduction to Data Communications*. Southern Alberta Institute of Technology, revision 2.1 ed., 2007. `http://www.telecomworld101.com/Intro2dcRev2/index.html`.

[12] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering, "IPv6 Flow Label Specification," RFC, RFC Editor, Mar. 2004.

[13] ITU-T Telecommunication standardization sector of ITU, "X.200 : Information technology - Open Systems Interconnection - Basic Reference Model: The basic model." `http://www.itu.int/rec/T-REC-X.200-199407-I/en`, July 1994.

[14] R. Braden, "Requirements for Internet Hosts - Communication Layers," STD, RFC Editor, Oct. 1989. `http://www.rfc-editor.org/rfc/rfc1122.txt`.

[15] J. F. Kurose and K. W. Ross, *Computer Networking : A Top-down Approach*. Global Edition, Boston München: Pearson Education Limited, global edition, seventh edition. ed., 2016.

[16] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, RFC Editor / RFC Editor, May 2015. `http://www.rfc-editor.org/rfc/rfc7540.txt`.

[17] J. Postel and J. Reynolds, "File Transfer Protocol," STD 9, RFC Editor / RFC Editor, Oct. 1985. `http://www.rfc-editor.org/rfc/rfc959.txt`.

[18] J. Postel, "Internet Protocol," STD 5, RFC Editor / RFC Editor, Sept. 1981. `http://www.rfc-editor.org/rfc/rfc791.txt`.

[19] S. E. Deering and R. M. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, RFC Editor / RFC Editor, Dec. 1998. `http://www.rfc-editor.org/rfc/rfc2460.txt`.

[20] W.-F. Alliance, "Wi-Fi CERTIFIED™ MAKES IT Wi-Fi," *printed from www. wi-fi. org/wifi-protected-setup*, vol. 1, 2006. `http://senza-fili.com/downloads/Wi-Fi_Certified_Makes_it_Wi-Fi.pdf`.

[21] W. W. Peterson and D. T. Brown, "Cyclic Codes for Error Detection," *Proceedings of the IRE*, vol. 49, pp. 228–235, Jan. 1961.

[22] T. B. Community, "The Bufferbloat Project." https://www.bufferbloat.net/projects/.

[23] G. Almes, S. Kalidindi, M. Zekauskas, and A. Morton, "A One-Way Delay Metric for IP Performance Metrics (IPPM)," STD, RFC Editor, Jan. 2016. https://tools.ietf.org/html/rfc7679.

[24] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[25] C. Demichelis and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)," RFC, RFC Editor, Nov. 2002. http://www.rfc-editor.org/rfc/rfc3393.txt.

[26] R. Stewart, "Stream Control Transmission Protocol," RFC 4960, RFC Editor, Sept. 2007. http://www.rfc-editor.org/rfc/rfc4960.txt.

[27] G. Almes, S. Kalidindi, M. Zekauskas, and A. Morton, "A One-Way Loss Metric for IP Performance Metrics (IPPM)," STD, RFC Editor, Jan. 2016. https://tools.ietf.org/html/rfc7680.

[28] J.-C. Laprie, *Dependability : basic concepts and terminology in English, French, German, Italian and Japanese*. Wien [u.a.]: Springer, 1992.

[29] K. Nichols, S. Blake, F. Baker, and D. L. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC, RFC Editor, Dec. 1998. http://www.rfc-editor.org/rfc/rfc2474.txt.

[30] 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, 1993.

[31] M. Médard and S. S. Lumetta, "Network reliability and fault tolerance," *Encyclopedia of Telecommunications*, 2003. http://onlinelibrary.wiley.com/doi/10.1002/0471219282.eot281/full.

[32] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia Streaming via TCP: An Analytic Performance Study," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, pp. 16:1–16:22, May 2008. http://doi.acm.org/10.1145/1352012.1352020.

[33] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and*

Systems for Video Technology, vol. 22, pp. 1649–1668, Dec. 2012. http://ieeexplore.ieee.org/document/6316136/.

[34] "The WebM Project | VP9 Video Codec Summary." https://www.webmproject.org/vp9/.

[35] R. M. Beuschel, *Video Compression Systems for Low-Latency Applications*. PhD thesis, Universität Ulm, Jan. 2010. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.427.8159&rep=rep1&type=pdf.

[36] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," STD, RFC Editor, July 2003. http://www.rfc-editor.org/rfc/rfc3550.txt.

[37] H. Schulzrinne, A. Rao, R. Lanphier, M. Westerlund, and M. Stiemerling, "Real-Time Streaming Protocol Version 2.0," RFC 7826, RFC Editor / RFC Editor, Dec. 2016.

[38] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261, RFC Editor / RFC Editor, June 2002. http://www.rfc-editor.org/rfc/rfc3261.txt.

[39] M. Handley, V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," RFC 4566, RFC Editor / RFC Editor, July 2006. http://www.rfc-editor.org/rfc/rfc4566.txt.

[40] J. Technologies, *Network Protocols Handbook*. Javvin Technologies, 2005.

[41] G. Carle and E. Biersack, "Survey of error recovery techniques for IP-based audio-visual multicast applications," *Network, IEEE*, vol. 11, pp. 24–36, Nov. 1997.

[42] B. K. Artur Janicki, "Packet Loss Concealment Algorithm for VoIP Transmission in Unreliable Networks," in *Proceedings of the 2008 Conference on New Trends in Multimedia and Network Information Systems*, (Amsterdam, The Netherlands, The Netherlands), pp. 23–33, IOS Press, 2008. http://dl.acm.org/citation.cfm?id=1565754.1565759.

[43] Cisco, "Understanding Delay in Packet Voice Networks." https://www.cisco.com/c/en/us/support/docs/voice/voice-quality/5125-delay-details.html.

[44] International Organization for Standardization and ISO/TC 22/SC 35 Lighting and visibility group, "ISO 16505:2015 Road vehicles – Ergonomic and performance aspects of Camera Monitor Systems – Requirements and test procedures," tech. rep., May 2015.

[45] ITU-T Telecommunication standardization sector of ITU, "G.109 : Definition of categories of speech transmission quality." https://www.itu.int/rec/T-REC-G.109-199909-I/en, Sept. 1999.

[46] ITU-T Telecommunication standardization sector of ITU, "G.114: One-way transmission time." https://www.itu.int/rec/T-REC-G.114-200305-I/en, May 2003.

[47] J. Mahdavi and V. Paxson, "IPPM Metrics for Measuring Connectivity," RFC, RFC Editor, Sept. 1999.

[48] M. C. John and W. S. C. Vincent, "Ultra high connectivity military networks," in *2010 - MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE*, pp. 1011–1018, Oct. 2010.

[49] B. M. Sousa, K. Pentikousis, and M. Curado, "Multihoming Management for Future Networks," *Mobile Networks and Applications*, vol. 16, pp. 505–517, Aug. 2011. http://link.springer.com/10.1007/s11036-011-0323-5.

[50] J. Networks, "EVPN Multihoming Overview." https://www.juniper.net/documentation/en_US/junos/topics/concept/evpn-bgp-multihoming-overview.html, Dec. 2017.

[51] A. S. Khatouni, M. A. Marsan, and M. Mellia, "Delay tolerant video upload from public vehicles," in *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference On*, pp. 213–218, IEEE, 2016. http://ieeexplore.ieee.org/abstract/document/7562074/.

[52] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[53] C. Perkins, O. Hodson, and V. Hardman, "A survey of packet loss recovery techniques for streaming audio," *IEEE Network*, vol. 12, pp. 40–48, Sept. 1998.

[54] A. Li, "RTP Payload Format for Generic Forward Error Correction," RFC, RFC Editor, Dec. 2007.

[55] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery," RFC, RFC Editor, Oct. 2007.

[56] S. Aust, J. O. Kim, P. Davis, A. Yamaguchi, and S. Obana, "Evaluation of Linux Bonding Features," in *2006 International Conference on Communication Technology*, pp. 1–6, Nov. 2006.

[57] "IEEE Standard for Local and metropolitan area networks - Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks - Amendment 22: Equal Cost Multiple Path (ECMP)," *IEEE Std 802.1Qbp-2014 (Amendment to IEEE Std 802.1Q-2011)*, pp. 1–127, Apr. 2014.

[58] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "The Locator/ID Separation Protocol (LISP)," RFC, RFC Editor, Jan. 2013. `http://www.rfc-editor.org/rfc/rfc6830.txt`.

[59] B. Feng, H. Zhang, H. Zhou, and S. Yu, "Locator/Identifier Split Networking: A Promising Future Internet Architecture," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2017.

[60] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2," internet-Draft, IETF Secretariat, July 2016. `http://www.ietf.org/internet-drafts/draft-hamilton-early-deployment-quic-00.txt`.

[61] J. Postel, "User Datagram Protocol," STD, RFC Editor, Aug. 1980. `http://www.rfc-editor.org/rfc/rfc768.txt`.

[62] J. Postel, "Transmission Control Protocol," STD, RFC Editor, Sept. 1981. `http://www.rfc-editor.org/rfc/rfc793.txt`.

[63] M. S. Islam, M. A. Kashem, W. H. Sadid, M. A. Rahman, M. N. Islam, and S. Anam, "TCP variants and network parameters: A comprehensive performance analysis," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, pp. 18–20, 2009. `https://www.researchgate.net/profile/Md_Nazrul_Islam6/publication/44259574_TCP_Variants_and_Network_Parameters_A_Comprehensive_Performance_Analysis/links/5539e18e0cf226723aba35b9.pdf`.

[64] Y.-N. Lien, "Communication Networks, Mobile Communications, and Mobile Computing." `http://www.cs.nccu.edu.tw/~lien/Lib/Image/TCP/tcpcwnd.jpg`, 2016.

[65] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC, RFC Editor, Sept. 2009. `http://www.rfc-editor.org/rfc/rfc5681.txt`.

[66] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC, RFC Editor, Mar. 2006. http://www.rfc-editor.org/rfc/rfc4340.txt.

[67] S. Lande and J. Liu, "Implementation of the Deadline-based Selective Retransmission For Maintaining Low End-to-End Delays," 2009. http://micsymposium.org/mics_2009_proceedings/mics2009_submission_8.pdf.

[68] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC, RFC Editor, Jan. 2013. http://www.rfc-editor.org/rfc/rfc6824.txt.

[69] C. Xu, H. Huang, H. Zhang, C. Xiong, and L. Zhu, "Multipath Transmission Control Protocol (MPTCP) Partial Reliability Extension," internet-Draft, IETF Secretariat, June 2017. http://www.ietf.org/internet-drafts/draft-xu-mptcp-prmp-04.txt.

[70] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration," RFC, RFC Editor, Sept. 2007.

[71] D. Farinacci, "Scaling the Internet routing table with Locator/ID Separation Protocol (LISP)." https://www.youtube.com/watch?v=QIGSMLrU4Xw#action=share, June 2007.

[72] M. Tuexen, I. Ruengeler, and R. Stewart, "SACK-IMMEDIATLY Extension for the Stream Control Transmission Protocol," RFC 7053, RFC Editor, Nov. 2013. https://tools.ietf.org/html/rfc7053.

[73] J. Nagle, "Congestion Control in IP/TCP Internetworks," RFC, RFC Editor, Jan. 1984.

[74] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension," RFC, RFC Editor, May 2004. http://www.rfc-editor.org/rfc/rfc3758.txt.

[75] M. Tuexen, R. Seggelmann, R. Stewart, and S. Loreto, "Additional Policies for the Partially Reliable Stream Control Transmission Protocol Extension," RFC, RFC Editor, Apr. 2015. http://www.rfc-editor.org/rfc/rfc7496.txt.

[76] M. Rajiullah, *Performance Analysis and Improvement of PR-SCTP in an Event Logging Context*. PhD thesis, Karlstads universitet, Karlstad, 2012.

[77] S. Fallon, P. Jacob, Y. Qiao, L. Murphy, E. Fallon, and A. Hanley, "SCTP switchover performance issues in WLAN environments," in *2008 5th IEEE Consumer Communications and Networking Conference*, pp. 564–568, IEEE, 2008. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4446429.

[78] Y. Nishida, P. Natarajan, A. L. Caro, P. D. Amer, and K. Nielsen, "SCTP-PF: A Quick Failover Algorithm for the Stream Control Transmission Protocol," RFC 7829, Apr. 2016. https://rfc-editor.org/rfc/rfc7829.txt.

[79] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming over Independent End-to-end Paths," *IEEE/ACM Trans. Netw.*, vol. 14, pp. 951–964, Oct. 2006. http://dx.doi.org/10.1109/TNET.2006.882843.

[80] R. Stewart, M. Tüxen, and P. Lei, *SCTP: What Is It, and How to Use It?* BSDCan, 2008. http://www.getthestuff.in/wp-content/uploads/2016/03/44_bsdcan_sctp.pdf.

[81] T. Dreibholz, M. Becke, J. Pulinthanath, and E. P. Rathgeb, "Implementation and Evaluation of Concurrent Multipath Transfer for SCTP in the INET Framework," in *Proceedings of the 3rd ACM/ICST International Workshop on OMNeT++*, (Torremolinos, Málaga/Spain), Mar. 2010. https://www.wiwi.uni-due.de/fileadmin/fileupload/I-TDR/SCTP/Paper/OMNeT__Workshop2010-SCTP.pdf.

[82] H. Adhari, T. Dreibholz, and M. Becke, "SCTP Socket API Extensions for Concurrent Multipath Transfer," tech. rep., 2016. https://tools.ietf.org/html/draft-dreibholz-tsvwg-sctpsocket-multipath-13.

[83] L. Magalhaes and R. Kravets, "Transport level mechanisms for bandwidth aggregation on mobile hosts," in *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pp. 165–171, Nov. 2001.

[84] P. Natarajan, N. Ekiz, E. Yilmaz, P. D. Amer, J. Iyengar, and R. Stewart, "Non-Renegable Selective Acknowledgments (NR-SACKs) for SCTP," in *IEEE International Conference on Network Protocols, 2008. ICNP 2008*, pp. 187–196, Oct. 2008.

[85] N. Ekiz, P. Amer, P. Natarajan, R. Stewart, and J. Iyengar, "Non-Renegable Selective Acknowledgements (NR-SACKs) for SCTP," internet-Draft, IETF Secretariat, Aug. 2011. http://www.ietf.org/internet-drafts/draft-natarajan-tsvwg-sctp-nrsack-08.txt.

[86] T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tuxen, "On the Use of Concurrent Multipath Transfer over Asymmetric Paths," in *2010 IEEE Global Telecommunications Conference (GLOBECOM 2010)*, pp. 1–6, Dec. 2010.

[87] H. Adhari, T. Dreibholz, M. Becke, E. P. Rathgeb, and M. Tüxen, "Evaluation of Concurrent Multipath Transfer over Dissimilar Paths," in *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA)*, pp. 708–714, Mar. 2011.

[88] J. D. Cavanaugh, "Protocol Overhead in IP/ATM Networks," *ATM Networks*, p. 11.

[89] "Ns-3, a discrete-event network simulator for Internet systems." `https://www.nsnam.org/`.

[90] "OMNeT++ Discrete Event Simulator." `https://omnetpp.org/`.

[91] "GStreamer: Open source multimedia framework." `https://gstreamer.freedesktop.org/`.

[92] "Usrsctp: A portable SCTP userland stack." `https://github.com/sctplab/usrsctp`, Dec. 2017.

[93] A. Keller, *Trace Control for Netem*. Technical report, Eidgenössische Technische Hochschule Zürich, Institut für Technische Informatik und Kommunikationsnetze, tcn. hypert. net/tcn. pdf, 2006. 19.3. 2012. 70, 2012. `http://pub.tik.ee.ethz.ch/students/2006-So/SA-2006-15.pdf`.

[94] "Lksctp/lksctp-tools - Linux Kernel Stream Control Transmission Protocol Tools." `http://lksctp.sourceforge.net/`.

[95] "OpenWebRTC." `https://www.openwebrtc.org/`.

[96] "Usrsctp: Author's version." `https://github.com/stfl/usrsctp`, June 2017.

[97] R. Stewart, M. Tuexen, K. Poon, P. Lei, and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)," RFC, RFC Editor, Dec. 2011.

[98] Q. D. Coninck and O. Bonaventure, "Multipath Extension for QUIC," Internet-Draft draft-deconinck-quic-multipath-00, Internet Engineering Task Force, Mar. 2018. `https://datatracker.ietf.org/doc/html/draft-deconinck-quic-multipath-00`.

[99] I. Lubashev, "Partially Reliable Streams for QUIC," Internet-Draft draft-lubashev-quic-partial-reliability-01, Internet Engineering Task Force, Jan. 2018. https://datatracker.ietf.org/doc/html/draft-lubashev-quic-partial-reliability-01.

[100] P. Tiesel, M. Palmer, B. Chandrasekaran, A. Feldmann, and J. Ott, "Considerations for Unreliable Streams in QUIC," Internet-Draft draft-tiesel-quic-unreliable-streams-01, IETF Secretariat, Oct. 2017. http://www.ietf.org/internet-drafts/draft-tiesel-quic-unreliable-streams-01.txt.

A Appendix

This Appendix includes the comprehensive tables of test results that have been omitted in the previous part because of space limitation.

Table 11 presents the results for the measurements with the SCTP version with modified congestion control and Table 12 shows the results with the SCTP version with regular congestion control.

The columns within these tables are: The configured setup, the configured link delay δ_e and loss rate ρ parameters of the experiment; the number of performed test runs ($|R|$); the expected DDR ($E(DDR)$); the DDR of all test runs of this experiment set combined; the standard deviation of the DDR results of this experiment set (σDDR); the difference between the expected and the actual DDR result (ΔDDR); the median of the TTD measurements ($T\hat{T}D$); the 99th quantile of the measured TTD ($q_{99}TTD$) and the expected as well as measured TO.

Measured Results With Modified Congestion Control

Setup	δ_e	ρ	$\|R\|$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
cmt	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	62ms	62ms	0.11	0.09
cmt	20ms	0.6%	10	1.000	1.0000	0.00000	0.00004	62ms	62ms	0.12	0.09
cmt	20ms	2.5%	10	0.999	0.9692	0.03778	-0.03014	62ms	144ms	0.14	0.10
cmt	20ms	6.2%	9	0.996	0.9024	0.04159	-0.09365	62ms	145ms	0.18	0.11
cmt	20ms	8.8%	10	0.992	0.9268	0.02195	-0.06558	62ms	145ms	0.20	0.17
cmt	20ms	12.5%	10	0.984	0.9080	0.03676	-0.07633	62ms	160ms	0.24	0.24
cmt	40ms	0.0%	4	1.000	1.0000	0.00000	0.00000	82ms	83ms	0.11	0.07
cmt	40ms	0.6%	10	1.000	0.9977	0.00089	-0.00224	82ms	83ms	0.12	0.08
cmt	40ms	2.5%	10	0.999	0.9858	0.01852	-0.01360	82ms	163ms	0.14	0.11
cmt	40ms	6.2%	9	0.996	0.9242	0.03148	-0.07191	82ms	164ms	0.18	0.19
cmt	40ms	8.8%	10	0.992	0.9021	0.03080	-0.09024	82ms	164ms	0.20	0.23
cmt	40ms	12.5%	10	0.984	0.8783	0.02514	-0.10603	82ms	164ms	0.24	0.32
cmt	60ms	0.0%	8	1.000	1.0000	0.00000	0.00000	103ms	103ms	0.13	0.06
cmt	60ms	0.6%	10	0.994	0.9933	0.00080	-0.00046	102ms	103ms	0.13	0.08
cmt	60ms	2.5%	8	0.975	0.9752	0.00273	0.00022	102ms	103ms	0.13	0.10

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
cmt	60ms	6.2%	8	0.938	0.9366	0.00173	-0.00089	102ms	103ms	0.13	0.14
cmt	60ms	8.8%	10	0.912	0.9153	0.00646	0.00281	102ms	103ms	0.13	0.16
cmt	60ms	12.5%	10	0.875	0.8842	0.00767	0.00920	102ms	103ms	0.12	0.18
cmt	80ms	0.0%	6	1.000	1.0000	0.00000	0.00000	123ms	123ms	0.13	0.06
cmt	80ms	0.6%	10	0.994	0.9944	0.00144	0.00067	122ms	123ms	0.13	0.07
cmt	80ms	2.5%	9	0.975	0.9744	0.00265	-0.00060	122ms	123ms	0.13	0.12
cmt	80ms	6.2%	7	0.938	0.9384	0.00571	0.00091	122ms	123ms	0.13	0.16
cmt	80ms	8.8%	8	0.912	0.9133	0.00601	0.00082	122ms	123ms	0.13	0.17
cmt	80ms	12.5%	10	0.875	0.8815	0.00511	0.00646	122ms	123ms	0.12	0.17
cmt	100ms	0.0%	8	1.000	1.0000	0.00000	0.00000	142ms	143ms	0.13	0.05
cmt	100ms	0.6%	10	0.994	0.9940	0.00145	0.00025	142ms	143ms	0.13	0.08
cmt	100ms	2.5%	10	0.975	0.9730	0.00261	-0.00197	142ms	143ms	0.13	0.13
cmt	100ms	6.2%	10	0.938	0.9364	0.00375	-0.00114	142ms	143ms	0.13	0.16
cmt	100ms	8.8%	10	0.912	0.9169	0.00502	0.00443	142ms	143ms	0.13	0.17
cmt	100ms	12.5%	10	0.875	0.8840	0.00775	0.00903	142ms	143ms	0.12	0.18
dpr	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	62ms	63ms	0.11	0.09
dpr	20ms	0.6%	10	1.000	0.9947	0.01682	-0.00530	62ms	63ms	0.12	0.09
dpr	20ms	2.5%	10	0.999	0.9732	0.02759	-0.02622	62ms	144ms	0.14	0.11
dpr	20ms	6.2%	10	0.996	0.9356	0.01850	-0.06048	62ms	145ms	0.18	0.15
dpr	20ms	8.8%	9	0.992	0.9270	0.02434	-0.06532	62ms	145ms	0.20	0.19
dpr	20ms	12.5%	7	0.984	0.9150	0.04384	-0.06941	62ms	175ms	0.24	0.22
dpr	40ms	0.0%	4	1.000	1.0000	0.00000	0.00000	83ms	83ms	0.11	0.07
dpr	40ms	0.6%	10	1.000	0.9992	0.00039	-0.00073	82ms	83ms	0.12	0.08
dpr	40ms	2.5%	10	0.999	0.9815	0.02410	-0.01792	82ms	164ms	0.14	0.12
dpr	40ms	6.2%	10	0.996	0.9426	0.04061	-0.05349	82ms	192ms	0.18	0.19
dpr	40ms	8.8%	10	0.992	0.8786	0.06009	-0.11376	82ms	192ms	0.20	0.21
dpr	40ms	12.5%	10	0.984	0.9074	0.04151	-0.07697	82ms	193ms	0.24	0.32
dpr	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	102ms	103ms	1.18	0.64
dpr	60ms	0.6%	10	1.000	0.9975	0.00064	-0.00247	102ms	103ms	1.18	0.66
dpr	60ms	2.5%	10	0.999	0.9874	0.00226	-0.01194	102ms	192ms	1.18	0.69
dpr	60ms	6.2%	8	0.996	0.9686	0.00368	-0.02752	102ms	193ms	1.18	0.73

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
dpr	60ms	8.8%	10	0.992	0.9550	0.00435	-0.03731	102ms	194ms	1.18	0.73
dpr	60ms	12.5%	9	0.984	0.9314	0.02686	-0.05298	102ms	194ms	1.18	0.73
dpr	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	122ms	123ms	1.18	1.12
dpr	80ms	0.6%	10	1.000	0.9963	0.00071	-0.00362	122ms	123ms	1.18	1.13
dpr	80ms	2.5%	10	0.999	0.9860	0.00116	-0.01338	122ms	231ms	1.18	1.16
dpr	80ms	6.2%	10	0.996	0.9662	0.00314	-0.02986	122ms	234ms	1.18	1.17
dpr	80ms	8.8%	10	0.992	0.9542	0.00530	-0.03816	122ms	235ms	1.18	1.21
dpr	80ms	12.5%	9	0.984	0.9345	0.00456	-0.04985	122ms	235ms	1.18	1.24
dpr	100ms	0.0%	10	1.000	1.0000	0.00000	0.00000	142ms	143ms	1.18	1.11
dpr	100ms	0.6%	10	1.000	0.9960	0.00085	-0.00392	142ms	143ms	1.18	1.12
dpr	100ms	2.5%	10	0.999	0.9851	0.00206	-0.01424	142ms	231ms	1.18	1.15
dpr	100ms	6.2%	10	0.996	0.9625	0.00286	-0.03361	142ms	234ms	1.18	1.19
dpr	100ms	8.8%	10	0.992	0.9484	0.00386	-0.04399	142ms	234ms	1.18	1.20
dpr	100ms	12.5%	10	0.984	0.9138	0.04765	-0.07054	142ms	234ms	1.18	1.20
dpr_acc	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	62ms	63ms	0.11	0.09
dpr_acc	20ms	0.6%	10	1.000	0.9947	0.01682	-0.00530	62ms	63ms	0.12	0.09
dpr_acc	20ms	2.5%	10	0.999	0.9732	0.02759	-0.02622	62ms	144ms	0.14	0.11
dpr_acc	20ms	6.2%	10	0.996	0.9356	0.01850	-0.06045	62ms	145ms	0.18	0.15
dpr_acc	20ms	8.8%	9	0.992	0.9273	0.02434	-0.06504	62ms	145ms	0.20	0.19
dpr_acc	20ms	12.5%	7	0.984	0.9156	0.04384	-0.06881	62ms	175ms	0.24	0.22
dpr_acc	40ms	0.0%	4	1.000	1.0000	0.00000	0.00000	83ms	83ms	0.11	0.07
dpr_acc	40ms	0.6%	10	1.000	0.9992	0.00039	-0.00073	82ms	83ms	0.12	0.08
dpr_acc	40ms	2.5%	10	0.999	0.9817	0.02410	-0.01768	82ms	164ms	0.14	0.12
dpr_acc	40ms	6.2%	10	0.996	0.9429	0.04061	-0.05324	82ms	192ms	0.18	0.19
dpr_acc	40ms	8.8%	10	0.992	0.8788	0.06009	-0.11359	82ms	192ms	0.20	0.21
dpr_acc	40ms	12.5%	10	0.984	0.9079	0.04151	-0.07652	82ms	193ms	0.24	0.32
dpr_acc	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	102ms	103ms	1.18	0.64
dpr_acc	60ms	0.6%	10	1.000	0.9977	0.00064	-0.00230	102ms	103ms	1.18	0.66
dpr_acc	60ms	2.5%	10	0.999	0.9886	0.00226	-0.01079	102ms	192ms	1.18	0.69
dpr_acc	60ms	6.2%	8	0.996	0.9719	0.00368	-0.02420	102ms	193ms	1.18	0.73
dpr_acc	60ms	8.8%	10	0.992	0.9599	0.00435	-0.03243	102ms	194ms	1.18	0.73

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
dpr_acc	60ms	12.5%	9	0.984	0.9373	0.02686	-0.04704	102ms	194ms	1.18	0.73
dpr_acc	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	122ms	123ms	1.18	1.12
dpr_acc	80ms	0.6%	10	1.000	0.9999	0.00071	-0.00005	122ms	123ms	1.18	1.13
dpr_acc	80ms	2.5%	10	0.999	0.9991	0.00116	-0.00023	122ms	231ms	1.18	1.16
dpr_acc	80ms	6.2%	10	0.996	0.9962	0.00314	0.00014	122ms	234ms	1.18	1.17
dpr_acc	80ms	8.8%	10	0.992	0.9921	0.00530	-0.00024	122ms	235ms	1.18	1.21
dpr_acc	80ms	12.5%	9	0.984	0.9865	0.00456	0.00216	122ms	235ms	1.18	1.24
dpr_acc	100ms	0.0%	10	1.000	1.0000	0.00000	0.00000	142ms	143ms	1.18	1.11
dpr_acc	100ms	0.6%	10	1.000	1.0000	0.00085	0.00002	142ms	143ms	1.18	1.12
dpr_acc	100ms	2.5%	10	0.999	0.9994	0.00206	-0.00001	142ms	231ms	1.18	1.15
dpr_acc	100ms	6.2%	10	0.996	0.9957	0.00286	-0.00039	142ms	234ms	1.18	1.19
dpr_acc	100ms	8.8%	10	0.992	0.9918	0.00386	-0.00050	142ms	234ms	1.18	1.20
dpr_acc	100ms	12.5%	10	0.984	0.9715	0.04765	-0.01284	142ms	234ms	1.18	1.20
single	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	63ms	63ms	0.11	0.07
single	20ms	0.6%	10	1.000	1.0000	0.00000	0.00004	63ms	63ms	0.12	0.08
single	20ms	2.5%	10	0.999	1.0000	0.00000	0.00062	63ms	105ms	0.14	0.10
single	20ms	6.2%	9	0.996	1.0000	0.00000	0.00391	63ms	145ms	0.18	0.16
single	20ms	8.8%	10	0.992	0.9995	0.00027	0.00715	63ms	145ms	0.20	0.19
single	20ms	12.5%	10	0.984	0.9993	0.00034	0.01490	63ms	145ms	0.24	0.25
single	40ms	0.0%	10	1.000	1.0000	0.00000	0.00000	83ms	83ms	0.11	0.07
single	40ms	0.6%	10	1.000	0.9979	0.00069	-0.00203	83ms	83ms	0.12	0.08
single	40ms	2.5%	10	0.999	0.9916	0.00149	-0.00776	83ms	163ms	0.14	0.13
single	40ms	6.2%	10	0.996	0.9787	0.00235	-0.01743	83ms	164ms	0.18	0.19
single	40ms	8.8%	10	0.992	0.9679	0.00387	-0.02443	83ms	164ms	0.20	0.22
single	40ms	12.5%	9	0.984	0.9524	0.00307	-0.03195	82ms	164ms	0.24	0.26
single	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	103ms	103ms	0.13	0.06
single	60ms	0.6%	10	0.994	0.9934	0.00173	-0.00037	103ms	103ms	0.13	0.08
single	60ms	2.5%	9	0.975	0.9738	0.00241	-0.00119	103ms	103ms	0.13	0.11
single	60ms	6.2%	8	0.938	0.9381	0.00316	0.00057	103ms	103ms	0.13	0.15
single	60ms	8.8%	9	0.912	0.9138	0.00477	0.00125	102ms	103ms	0.13	0.16
single	60ms	12.5%	10	0.875	0.8795	0.00714	0.00445	102ms	103ms	0.12	0.17

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
single	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	123ms	123ms	0.13	0.06
single	80ms	0.6%	10	0.994	0.9943	0.00113	0.00057	123ms	123ms	0.13	0.08
single	80ms	2.5%	10	0.975	0.9732	0.00302	-0.00178	123ms	123ms	0.13	0.12
single	80ms	6.2%	10	0.938	0.9367	0.00425	-0.00084	122ms	123ms	0.13	0.16
single	80ms	8.8%	9	0.912	0.9130	0.00367	0.00054	122ms	123ms	0.13	0.17
single	80ms	12.5%	10	0.875	0.8793	0.00562	0.00433	122ms	123ms	0.12	0.17
single	100ms	0.0%	10	1.000	1.0000	0.00000	0.00000	143ms	143ms	0.13	0.05
single	100ms	0.6%	10	0.994	0.9941	0.00207	0.00031	143ms	143ms	0.13	0.08
single	100ms	2.5%	10	0.975	0.9738	0.00193	-0.00119	142ms	143ms	0.13	0.13
single	100ms	6.2%	10	0.938	0.9393	0.00457	0.00178	142ms	143ms	0.13	0.16
single	100ms	8.8%	10	0.912	0.9146	0.00615	0.00215	141ms	143ms	0.13	0.17
single	100ms	12.5%	10	0.875	0.8828	0.00655	0.00782	141ms	142ms	0.12	0.17
udp	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	63ms	63ms	0.04	0.03
udp	20ms	0.6%	10	0.994	0.9932	0.00217	-0.00056	63ms	63ms	0.04	0.03
udp	20ms	2.5%	10	0.975	0.9749	0.00285	-0.00010	63ms	63ms	0.04	0.03
udp	20ms	6.2%	10	0.938	0.9380	0.00503	0.00055	63ms	63ms	0.04	0.03
udp	20ms	8.8%	10	0.912	0.9156	0.00451	0.00311	63ms	63ms	0.04	0.04
udp	20ms	12.5%	10	0.875	0.8802	0.00591	0.00516	63ms	63ms	0.04	0.04
udp	40ms	0.0%	10	1.000	1.0000	0.00000	0.00000	83ms	83ms	0.04	0.03
udp	40ms	0.6%	10	0.994	0.9934	0.00120	-0.00039	83ms	83ms	0.04	0.03
udp	40ms	2.5%	10	0.975	0.9722	0.00308	-0.00280	83ms	83ms	0.04	0.02
udp	40ms	6.2%	10	0.938	0.9362	0.00519	-0.00126	83ms	83ms	0.04	0.03
udp	40ms	8.8%	10	0.912	0.9143	0.00592	0.00183	83ms	83ms	0.04	0.04
udp	40ms	12.5%	10	0.875	0.8800	0.00379	0.00499	83ms	83ms	0.04	0.04
udp	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	103ms	103ms	0.04	0.03
udp	60ms	0.6%	10	0.994	0.9934	0.00192	-0.00039	103ms	103ms	0.04	0.03
udp	60ms	2.5%	10	0.975	0.9742	0.00255	-0.00085	103ms	103ms	0.04	0.03
udp	60ms	6.2%	10	0.938	0.9360	0.00272	-0.00150	103ms	103ms	0.04	0.03
udp	60ms	8.8%	10	0.912	0.9132	0.00672	0.00074	103ms	103ms	0.04	0.03
udp	60ms	12.5%	10	0.875	0.8828	0.00554	0.00780	103ms	103ms	0.04	0.05
udp	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	123ms	123ms	0.04	0.03

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
udp	80ms	0.6%	10	0.994	0.9932	0.00153	-0.00052	123ms	123ms	0.04	0.03
udp	80ms	2.5%	10	0.975	0.9751	0.00389	0.00007	123ms	123ms	0.04	0.03
udp	80ms	6.2%	10	0.938	0.9380	0.00249	0.00050	123ms	123ms	0.04	0.03
udp	80ms	8.8%	10	0.912	0.9156	0.00412	0.00308	123ms	123ms	0.04	0.03
udp	80ms	12.5%	9	0.875	0.8843	0.00462	0.00930	123ms	123ms	0.04	0.05
udp	100ms	0.0%	10	1.000	1.0000	0.00000	0.00000	143ms	143ms	0.04	0.03
udp	100ms	0.6%	10	0.994	0.9932	0.00144	-0.00058	143ms	143ms	0.04	0.03
udp	100ms	2.5%	10	0.975	0.9745	0.00236	-0.00051	143ms	143ms	0.04	0.03
udp	100ms	6.2%	10	0.938	0.9391	0.00202	0.00159	143ms	143ms	0.04	0.03
udp	100ms	8.8%	10	0.912	0.9143	0.00684	0.00183	143ms	143ms	0.04	0.03
udp	100ms	12.5%	10	0.875	0.8835	0.00522	0.00850	143ms	143ms	0.04	0.05
udpdupl	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	62ms	63ms	1.07	1.04
udpdupl	20ms	0.6%	10	1.000	0.9999	0.00015	-0.00005	62ms	63ms	1.07	1.04
udpdupl	20ms	2.5%	10	0.999	0.9995	0.00021	0.00011	62ms	63ms	1.07	1.04
udpdupl	20ms	6.2%	10	0.996	0.9965	0.00096	0.00037	62ms	63ms	1.07	1.05
udpdupl	20ms	8.8%	10	0.992	0.9928	0.00132	0.00050	62ms	63ms	1.07	1.06
udpdupl	20ms	12.5%	9	0.984	0.9861	0.00137	0.00169	62ms	63ms	1.07	1.07
udpdupl	40ms	0.0%	10	1.000	1.0000	0.00000	0.00000	82ms	83ms	1.07	1.04
udpdupl	40ms	0.6%	10	1.000	0.9999	0.00015	-0.00005	82ms	83ms	1.07	1.04
udpdupl	40ms	2.5%	10	0.999	0.9991	0.00041	-0.00025	82ms	83ms	1.07	1.04
udpdupl	40ms	6.2%	10	0.996	0.9964	0.00078	0.00027	82ms	83ms	1.07	1.05
udpdupl	40ms	8.8%	9	0.992	0.9925	0.00092	0.00011	82ms	83ms	1.07	1.05
udpdupl	40ms	12.5%	10	0.984	0.9858	0.00171	0.00138	82ms	83ms	1.07	1.07
udpdupl	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	102ms	103ms	1.07	1.04
udpdupl	60ms	0.6%	10	1.000	0.9999	0.00010	-0.00002	102ms	103ms	1.07	1.04
udpdupl	60ms	2.5%	10	0.999	0.9995	0.00025	0.00011	102ms	103ms	1.07	1.04
udpdupl	60ms	6.2%	10	0.996	0.9965	0.00092	0.00039	102ms	103ms	1.07	1.05
udpdupl	60ms	8.8%	9	0.992	0.9930	0.00152	0.00065	102ms	103ms	1.07	1.05
udpdupl	60ms	12.5%	10	0.984	0.9855	0.00141	0.00113	102ms	103ms	1.07	1.07
udpdupl	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	122ms	123ms	1.07	1.04
udpdupl	80ms	0.6%	10	1.000	0.9999	0.00010	-0.00002	122ms	123ms	1.07	1.04

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
udpdupl	80ms	2.5%	10	0.999	0.9992	0.00042	-0.00014	122ms	123ms	1.07	1.04
udpdupl	80ms	6.2%	10	0.996	0.9958	0.00108	-0.00033	122ms	123ms	1.07	1.05
udpdupl	80ms	8.8%	10	0.992	0.9930	0.00116	0.00061	122ms	123ms	1.07	1.05
udpdupl	80ms	12.5%	8	0.984	0.9865	0.00193	0.00216	122ms	123ms	1.07	1.07
udpdupl	100ms	0.0%	10	1.000	1.0000	0.00000	0.00000	142ms	143ms	1.07	1.04
udpdupl	100ms	0.6%	10	1.000	1.0000	0.00000	0.00004	142ms	143ms	1.07	1.04
udpdupl	100ms	2.5%	10	0.999	0.9993	0.00043	-0.00010	142ms	143ms	1.07	1.04
udpdupl	100ms	6.2%	10	0.996	0.9962	0.00061	0.00010	142ms	143ms	1.07	1.05
udpdupl	100ms	8.8%	10	0.992	0.9931	0.00157	0.00074	142ms	143ms	1.07	1.05
udpdupl	100ms	12.5%	10	0.984	0.9861	0.00198	0.00168	142ms	143ms	1.07	1.07

Table 11: Measured Results With Modified Congestion Control

Measured Results With Regular Congestion Control

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
cmt	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	62ms	63ms	0.11	0.09
cmt	20ms	0.6%	10	1.000	0.9943	0.01773	-0.00571	62ms	102ms	0.12	0.09
cmt	20ms	2.5%	10	0.999	0.9853	0.02377	-0.01407	62ms	176ms	0.14	0.11
cmt	20ms	6.2%	10	0.996	0.9577	0.00784	-0.03840	119ms	195ms	0.18	0.18
cmt	20ms	8.8%	10	0.992	0.9315	0.01247	-0.06081	143ms	196ms	0.20	0.21
cmt	20ms	12.5%	9	0.984	0.8547	0.06804	-0.12972	160ms	198ms	0.24	0.28
cmt	40ms	0.6%	9	1.000	0.9969	0.00099	-0.00301	82ms	136ms	0.12	0.08
cmt	40ms	2.5%	10	0.999	0.9794	0.00379	-0.01998	83ms	188ms	0.14	0.10
cmt	40ms	6.2%	8	0.996	0.9401	0.00670	-0.05603	117ms	195ms	0.18	0.12
cmt	40ms	8.8%	9	0.992	0.9163	0.00415	-0.07609	122ms	195ms	0.20	0.13
cmt	40ms	12.5%	10	0.984	0.8500	0.06622	-0.13441	137ms	195ms	0.24	0.17
cmt	60ms	0.0%	1	1.000	1.0000	0.00000	0.00000	102ms	103ms	0.13	0.06
cmt	60ms	2.5%	4	0.975	0.9708	0.00268	-0.00422	102ms	103ms	0.13	0.10
cmt	60ms	6.2%	6	0.938	0.9354	0.00313	-0.00212	102ms	150ms	0.13	0.12
cmt	60ms	8.8%	6	0.912	0.9171	0.00399	0.00465	102ms	103ms	0.13	0.14
cmt	60ms	12.5%	5	0.875	0.8833	0.00381	0.00833	102ms	103ms	0.12	0.15
cmt	80ms	0.6%	3	0.994	0.9945	0.00085	0.00071	122ms	123ms	0.13	0.07
cmt	80ms	2.5%	2	0.975	0.9698	0.00301	-0.00523	122ms	123ms	0.13	0.10
cmt	80ms	6.2%	4	0.938	0.9367	0.00298	-0.00084	122ms	123ms	0.13	0.13
cmt	80ms	8.8%	6	0.912	0.9143	0.00428	0.00177	122ms	142ms	0.13	0.13
cmt	80ms	12.5%	10	0.875	0.8809	0.00632	0.00588	122ms	124ms	0.12	0.14
cmt	100ms	6.2%	2	0.938	0.9394	0.00166	0.00193	142ms	143ms	0.13	0.12
cmt	100ms	12.5%	4	0.875	0.8851	0.00716	0.01009	142ms	143ms	0.12	0.13
dpr	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	62ms	63ms	0.11	0.09
dpr	20ms	0.6%	10	1.000	1.0000	0.00007	0.00002	63ms	102ms	0.12	0.09
dpr	20ms	2.5%	10	0.999	0.9768	0.04201	-0.02254	63ms	192ms	0.14	0.15
dpr	20ms	6.2%	10	0.996	0.9228	0.07759	-0.07331	140ms	195ms	0.18	0.46
dpr	20ms	8.8%	9	0.992	0.8801	0.06311	-0.11225	169ms	197ms	0.20	0.56
dpr	20ms	12.5%	10	0.984	0.8170	0.08386	-0.16734	180ms	231ms	0.24	0.74
dpr	40ms	0.0%	10	1.000	1.0000	0.00000	0.00000	82ms	83ms	0.11	0.10

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
dpr	40ms	0.6%	10	1.000	0.9975	0.00084	-0.00249	82ms	121ms	0.12	0.12
dpr	40ms	2.5%	9	0.999	0.9879	0.01073	-0.01151	82ms	192ms	0.14	0.24
dpr	40ms	6.2%	9	0.996	0.9439	0.05390	-0.05221	121ms	195ms	0.18	0.56
dpr	40ms	8.8%	9	0.992	0.9398	0.01630	-0.05257	141ms	230ms	0.20	0.68
dpr	40ms	12.5%	9	0.984	0.8974	0.04452	-0.08697	148ms	234ms	0.24	0.75
dpr	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	102ms	103ms	1.18	0.63
dpr	60ms	0.6%	10	1.000	0.9966	0.00100	-0.00335	102ms	103ms	1.18	0.66
dpr	60ms	2.5%	10	0.999	0.9880	0.00215	-0.01138	102ms	192ms	1.18	0.69
dpr	60ms	6.2%	9	0.996	0.9696	0.00315	-0.02652	102ms	193ms	1.18	0.73
dpr	60ms	8.8%	10	0.992	0.9555	0.00349	-0.03682	102ms	194ms	1.18	0.73
dpr	60ms	12.5%	10	0.984	0.9367	0.00438	-0.04763	102ms	194ms	1.18	0.74
dpr	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	122ms	123ms	1.18	1.10
dpr	80ms	0.6%	10	1.000	0.9966	0.00130	-0.00341	122ms	123ms	1.18	1.11
dpr	80ms	2.5%	10	0.999	0.9860	0.00209	-0.01338	123ms	231ms	1.18	1.14
dpr	80ms	6.2%	9	0.996	0.9505	0.04060	-0.04558	122ms	234ms	1.18	1.12
dpr	80ms	8.8%	10	0.992	0.9531	0.00421	-0.03922	122ms	234ms	1.18	1.19
dpr	80ms	12.5%	10	0.984	0.8188	0.22273	-0.16560	122ms	235ms	1.18	0.94
dpr	100ms	0.0%	10	1.000	1.0000	0.00000	0.00000	142ms	143ms	1.18	1.08
dpr	100ms	0.6%	9	1.000	0.9958	0.00113	-0.00420	142ms	143ms	1.18	1.09
dpr	100ms	2.5%	10	0.999	0.9398	0.14062	-0.05960	142ms	231ms	1.18	1.00
dpr	100ms	6.2%	9	0.996	0.9630	0.00474	-0.03314	142ms	234ms	1.18	1.16
dpr	100ms	8.8%	10	0.992	0.9049	0.11256	-0.08743	142ms	234ms	1.18	1.05
dpr	100ms	12.5%	10	0.984	0.6813	0.29387	-0.30303	142ms	234ms	1.18	0.63
dpr_acc	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	62ms	63ms	0.11	0.09
dpr_acc	20ms	0.6%	10	1.000	1.0000	0.00007	0.00002	63ms	102ms	0.12	0.09
dpr_acc	20ms	2.5%	10	0.999	0.9771	0.04201	-0.02226	63ms	192ms	0.14	0.15
dpr_acc	20ms	6.2%	10	0.996	0.9256	0.07759	-0.07052	140ms	195ms	0.18	0.46
dpr_acc	20ms	8.8%	9	0.992	0.8854	0.06311	-0.10693	169ms	197ms	0.20	0.56
dpr_acc	20ms	12.5%	10	0.984	0.8267	0.08386	-0.15768	180ms	231ms	0.24	0.74
dpr_acc	40ms	0.0%	10	1.000	1.0000	0.00000	0.00000	82ms	83ms	0.11	0.10
dpr_acc	40ms	0.6%	10	1.000	0.9975	0.00084	-0.00249	82ms	121ms	0.12	0.12

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
dpr_acc	40ms	2.5%	9	0.999	0.9885	0.01073	-0.01085	82ms	192ms	0.14	0.24
dpr_acc	40ms	6.2%	9	0.996	0.9504	0.05390	-0.04565	121ms	195ms	0.18	0.56
dpr_acc	40ms	8.8%	9	0.992	0.9495	0.01630	-0.04285	141ms	230ms	0.20	0.68
dpr_acc	40ms	12.5%	9	0.984	0.9164	0.04452	-0.06795	148ms	234ms	0.24	0.75
dpr_acc	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	102ms	103ms	1.18	0.63
dpr_acc	60ms	0.6%	10	1.000	0.9971	0.00100	-0.00286	102ms	103ms	1.18	0.66
dpr_acc	60ms	2.5%	10	0.999	0.9895	0.00215	-0.00991	102ms	192ms	1.18	0.69
dpr_acc	60ms	6.2%	9	0.996	0.9727	0.00315	-0.02335	102ms	193ms	1.18	0.73
dpr_acc	60ms	8.8%	10	0.992	0.9605	0.00349	-0.03184	102ms	194ms	1.18	0.73
dpr_acc	60ms	12.5%	10	0.984	0.9430	0.00438	-0.04141	102ms	194ms	1.18	0.74
dpr_acc	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	122ms	123ms	1.18	1.10
dpr_acc	80ms	0.6%	10	1.000	1.0000	0.00130	-0.00000	122ms	123ms	1.18	1.11
dpr_acc	80ms	2.5%	10	0.999	0.9995	0.00209	0.00014	123ms	231ms	1.18	1.14
dpr_acc	80ms	6.2%	9	0.996	0.9835	0.04060	-0.01261	122ms	234ms	1.18	1.12
dpr_acc	80ms	8.8%	10	0.992	0.9929	0.00421	0.00057	122ms	234ms	1.18	1.19
dpr_acc	80ms	12.5%	10	0.984	0.8659	0.22273	-0.11848	122ms	235ms	1.18	0.94
dpr_acc	100ms	0.0%	10	1.000	1.0000	0.00000	0.00000	142ms	143ms	1.18	1.08
dpr_acc	100ms	0.6%	9	1.000	0.9999	0.00113	-0.00008	142ms	143ms	1.18	1.09
dpr_acc	100ms	2.5%	10	0.999	0.9538	0.14062	-0.04562	142ms	231ms	1.18	1.00
dpr_acc	100ms	6.2%	9	0.996	0.9957	0.00474	-0.00035	142ms	234ms	1.18	1.16
dpr_acc	100ms	8.8%	10	0.992	0.9472	0.11256	-0.04516	142ms	234ms	1.18	1.05
dpr_acc	100ms	12.5%	10	0.984	0.7267	0.29387	-0.25763	142ms	234ms	1.18	0.63
single	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	62ms	63ms	0.11	0.07
single	20ms	0.6%	10	1.000	0.9989	0.00069	-0.00107	63ms	182ms	0.12	0.09
single	20ms	2.5%	10	0.999	0.9918	0.00300	-0.00757	99ms	187ms	0.14	0.14
single	20ms	6.2%	9	0.996	0.9629	0.00669	-0.03316	134ms	197ms	0.18	0.17
single	20ms	8.8%	10	0.992	0.9435	0.00412	-0.04887	143ms	198ms	0.20	0.18
single	20ms	12.5%	10	0.984	0.9018	0.00608	-0.08259	157ms	198ms	0.24	0.14
single	40ms	0.0%	10	1.000	1.0000	0.00000	0.00000	83ms	83ms	0.11	0.06
single	40ms	0.6%	10	1.000	0.9975	0.00084	-0.00249	83ms	152ms	0.12	0.07
single	40ms	2.5%	9	0.999	0.9839	0.00215	-0.01544	97ms	191ms	0.14	0.10

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
single	40ms	6.2%	8	0.996	0.9435	0.00457	-0.05262	129ms	195ms	0.18	0.11
single	40ms	8.8%	9	0.992	0.9203	0.00272	-0.07209	133ms	196ms	0.20	0.10
single	40ms	12.5%	9	0.984	0.8861	0.00748	-0.09828	137ms	196ms	0.24	0.11
single	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	102ms	103ms	0.13	0.03
single	60ms	0.6%	10	0.994	0.9938	0.00108	0.00003	102ms	103ms	0.13	0.05
single	60ms	2.5%	8	0.975	0.9736	0.00267	-0.00140	102ms	103ms	0.13	0.08
single	60ms	6.2%	10	0.938	0.9380	0.00542	0.00055	102ms	103ms	0.13	0.12
single	60ms	8.8%	10	0.912	0.9126	0.00706	0.00015	102ms	103ms	0.13	0.13
single	60ms	12.5%	10	0.875	0.8819	0.00507	0.00690	102ms	103ms	0.12	0.14
single	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	122ms	129ms	0.13	-0.00
single	80ms	0.6%	10	0.994	0.9941	0.00073	0.00031	122ms	123ms	0.13	0.02
single	80ms	2.5%	10	0.975	0.9736	0.00226	-0.00144	122ms	123ms	0.13	0.06
single	80ms	6.2%	6	0.938	0.9360	0.00444	-0.00148	122ms	123ms	0.13	0.10
single	80ms	8.8%	8	0.912	0.9123	0.00734	-0.00024	122ms	123ms	0.13	0.11
single	80ms	12.5%	10	0.875	0.8823	0.00459	0.00729	122ms	123ms	0.12	0.13
single	100ms	0.0%	10	1.000	0.9918	0.00114	-0.00815	142ms	162ms	0.13	-0.03
single	100ms	0.6%	10	0.994	0.9879	0.00219	-0.00584	142ms	161ms	0.13	-0.01
single	100ms	2.5%	10	0.975	0.9576	0.03942	-0.01741	142ms	177ms	0.13	0.02
single	100ms	6.2%	10	0.938	0.9138	0.04710	-0.02366	142ms	183ms	0.13	0.04
single	100ms	8.8%	10	0.912	0.9133	0.00576	0.00081	142ms	162ms	0.13	0.08
single	100ms	12.5%	10	0.875	0.8802	0.00470	0.00516	142ms	151ms	0.12	0.09
udp	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	63ms	63ms	0.04	0.03
udp	20ms	0.6%	10	0.994	0.9932	0.00217	-0.00056	63ms	63ms	0.04	0.03
udp	20ms	2.5%	10	0.975	0.9749	0.00285	-0.00010	63ms	63ms	0.04	0.03
udp	20ms	6.2%	10	0.938	0.9380	0.00503	0.00055	63ms	63ms	0.04	0.03
udp	20ms	8.8%	10	0.912	0.9156	0.00451	0.00311	63ms	63ms	0.04	0.04
udp	20ms	12.5%	10	0.875	0.8802	0.00591	0.00516	63ms	63ms	0.04	0.04
udp	40ms	0.0%	10	1.000	1.0000	0.00000	0.00000	83ms	83ms	0.04	0.03
udp	40ms	0.6%	10	0.994	0.9934	0.00120	-0.00039	83ms	83ms	0.04	0.03
udp	40ms	2.5%	10	0.975	0.9722	0.00308	-0.00280	83ms	83ms	0.04	0.02
udp	40ms	6.2%	10	0.938	0.9362	0.00519	-0.00126	83ms	83ms	0.04	0.03

Setup	δ_e	ρ	$\|R\|$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
udp	40ms	8.8%	10	0.912	0.9143	0.00592	0.00183	83ms	83ms	0.04	0.04
udp	40ms	12.5%	10	0.875	0.8800	0.00379	0.00499	83ms	83ms	0.04	0.04
udp	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	103ms	103ms	0.04	0.03
udp	60ms	0.6%	10	0.994	0.9934	0.00192	-0.00039	103ms	103ms	0.04	0.03
udp	60ms	2.5%	10	0.975	0.9742	0.00255	-0.00085	103ms	103ms	0.04	0.03
udp	60ms	6.2%	10	0.938	0.9360	0.00272	-0.00150	103ms	103ms	0.04	0.03
udp	60ms	8.8%	10	0.912	0.9132	0.00672	0.00074	103ms	103ms	0.04	0.03
udp	60ms	12.5%	10	0.875	0.8828	0.00554	0.00780	103ms	103ms	0.04	0.05
udp	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	123ms	123ms	0.04	0.03
udp	80ms	0.6%	10	0.994	0.9932	0.00153	-0.00052	123ms	123ms	0.04	0.03
udp	80ms	2.5%	10	0.975	0.9751	0.00389	0.00007	123ms	123ms	0.04	0.03
udp	80ms	6.2%	10	0.938	0.9380	0.00249	0.00050	123ms	123ms	0.04	0.03
udp	80ms	8.8%	10	0.912	0.9156	0.00412	0.00308	123ms	123ms	0.04	0.03
udp	80ms	12.5%	9	0.875	0.8843	0.00462	0.00930	123ms	123ms	0.04	0.05
udp	100ms	0.0%	10	1.000	1.0000	0.00000	0.00000	143ms	143ms	0.04	0.03
udp	100ms	0.6%	10	0.994	0.9932	0.00144	-0.00058	143ms	143ms	0.04	0.03
udp	100ms	2.5%	10	0.975	0.9745	0.00236	-0.00051	143ms	143ms	0.04	0.03
udp	100ms	6.2%	10	0.938	0.9391	0.00202	0.00159	143ms	143ms	0.04	0.03
udp	100ms	8.8%	10	0.912	0.9143	0.00684	0.00183	143ms	143ms	0.04	0.03
udp	100ms	12.5%	10	0.875	0.8835	0.00522	0.00850	143ms	143ms	0.04	0.05
udpdupl	20ms	0.0%	10	1.000	1.0000	0.00000	0.00000	62ms	63ms	1.07	1.04
udpdupl	20ms	0.6%	10	1.000	0.9999	0.00015	-0.00005	62ms	63ms	1.07	1.04
udpdupl	20ms	2.5%	10	0.999	0.9995	0.00021	0.00011	62ms	63ms	1.07	1.04
udpdupl	20ms	6.2%	10	0.996	0.9965	0.00096	0.00037	62ms	63ms	1.07	1.05
udpdupl	20ms	8.8%	10	0.992	0.9928	0.00132	0.00050	62ms	63ms	1.07	1.06
udpdupl	20ms	12.5%	9	0.984	0.9861	0.00137	0.00169	62ms	63ms	1.07	1.07
udpdupl	40ms	0.0%	10	1.000	1.0000	0.00000	0.00000	82ms	83ms	1.07	1.04
udpdupl	40ms	0.6%	10	1.000	0.9999	0.00015	-0.00005	82ms	83ms	1.07	1.04
udpdupl	40ms	2.5%	10	0.999	0.9991	0.00041	-0.00025	82ms	83ms	1.07	1.04
udpdupl	40ms	6.2%	10	0.996	0.9964	0.00078	0.00027	82ms	83ms	1.07	1.05
udpdupl	40ms	8.8%	9	0.992	0.9925	0.00092	0.00011	82ms	83ms	1.07	1.05

Setup	δ_e	ρ	$\lvert R \rvert$	$E(DDR)$	DDR	σDDR	ΔDDR	$T\hat{T}D$	$q_{99}TTD$	$E(TO)$	TO
udpdupl	40ms	12.5%	10	0.984	0.9858	0.00171	0.00138	82ms	83ms	1.07	1.07
udpdupl	60ms	0.0%	10	1.000	1.0000	0.00000	0.00000	102ms	103ms	1.07	1.04
udpdupl	60ms	0.6%	10	1.000	0.9999	0.00010	-0.00002	102ms	103ms	1.07	1.04
udpdupl	60ms	2.5%	10	0.999	0.9995	0.00025	0.00011	102ms	103ms	1.07	1.04
udpdupl	60ms	6.2%	10	0.996	0.9965	0.00092	0.00039	102ms	103ms	1.07	1.05
udpdupl	60ms	8.8%	9	0.992	0.9930	0.00152	0.00065	102ms	103ms	1.07	1.05
udpdupl	60ms	12.5%	10	0.984	0.9855	0.00141	0.00113	102ms	103ms	1.07	1.07
udpdupl	80ms	0.0%	10	1.000	1.0000	0.00000	0.00000	122ms	123ms	1.07	1.04
udpdupl	80ms	0.6%	10	1.000	0.9999	0.00010	-0.00002	122ms	123ms	1.07	1.04
udpdupl	80ms	2.5%	10	0.999	0.9992	0.00042	-0.00014	122ms	123ms	1.07	1.04
udpdupl	80ms	6.2%	10	0.996	0.9958	0.00108	-0.00033	122ms	123ms	1.07	1.05
udpdupl	80ms	8.8%	10	0.992	0.9930	0.00116	0.00061	122ms	123ms	1.07	1.05
udpdupl	80ms	12.5%	8	0.984	0.9865	0.00193	0.00216	122ms	123ms	1.07	1.07
udpdupl	100ms	0.0%	10	1.000	1.0000	0.00000	0.00000	142ms	143ms	1.07	1.04
udpdupl	100ms	0.6%	10	1.000	1.0000	0.00000	0.00004	142ms	143ms	1.07	1.04
udpdupl	100ms	2.5%	10	0.999	0.9993	0.00043	-0.00010	142ms	143ms	1.07	1.04
udpdupl	100ms	6.2%	10	0.996	0.9962	0.00061	0.00010	142ms	143ms	1.07	1.05
udpdupl	100ms	8.8%	10	0.992	0.9931	0.00157	0.00074	142ms	143ms	1.07	1.05
udpdupl	100ms	12.5%	10	0.984	0.9861	0.00198	0.00168	142ms	143ms	1.07	1.07

Table 12: Measured Results With Regular Congestion Control

Setup	δ_e	$\lvert R \rvert$	PFI
cmt	20ms	10	0.4919
cmt	40ms	10	0.4429
cmt	60ms	10	0.3503
cmt	80ms	10	0.3457
cmt	100ms	10	0.3385
dpr	20ms	10	0.8370
dpr	40ms	9	0.7588
dpr	60ms	10	0.7460
dpr	80ms	9	0.6907
dpr	100ms	8	0.6520
dpr_acc	20ms	10	0.8381
dpr_acc	40ms	9	0.8594
dpr_acc	60ms	10	0.9583
dpr_acc	80ms	9	0.9848
dpr_acc	100ms	8	0.9347
single	20ms	10	0.0264
single	40ms	10	0.0292
single	60ms	10	0.0278
single	80ms	10	0.0361
single	100ms	10	0.0278
udp	20ms	10	0.0181
udp	40ms	10	0.0278
udp	60ms	10	0.0264
udp	80ms	10	0.0264
udp	100ms	10	0.0278
udpdupl	20ms	10	0.9917
udpdupl	40ms	10	0.9861
udpdupl	60ms	10	0.9861
udpdupl	80ms	10	0.9917
udpdupl	100ms	10	0.9944

Table 13: PFI measurement results

Table of Abbreviations

ACK	Acknowledgment
API	Application Programming Interface
BBR	Bottleneck Bandwidth and Round-trip propagation time
BDP	Bandwidth Delay Product
CDF	Cumulative Distribution Function
CMT	Concurrent Multipath Transmission
CRC	Cyclic Redundancy Check
CWND	Congestion Window
CumACK	Cumulative Acknowledgment
DAPS	Delay-Aware Packet Scheduling
DAR	Dynamic Address Reconfiguration
DCCP	Datagram Congestion Control Protocol
DDR	Delivery Within Deadline Ratio
DNS	Domain Name System
DPR	Deadline Based Preemptive Retransmission
DiffServ	Differentiated Services
ECMP	Equal Cost Multi Path
FEC	Forward Error Correction
FPD	Full Packet Duplication
FRTX	Fast Retransmission
FTP	File Transfer Protocol
GLBP	Gateway Load Balancing Protocol
GPS	Global Positioning System
HSPA	High Speed Packet Access
HSRP	Hot Standby Router Protocol
HTTP	Hypertext Transfer Protocol

IPPM	IP Performance Metrics
IPv4	IP Version 4
IPv6	IP Version 6
IP	Internet Protocol
ISP	Internet Service Provider
ITU	International Telecommunication Union
LC	Link Capacity
LISP	Locator/Identifier Split Protocol
LTE	Long Term Evolution
MAC	Media Access Control
MAC	Message authentication code
MP-TCP	Multipath TCP
MTU	Maximum Transfer Unit
NR-SACK	Non-Renegable Selective Acknowledgments
NTP	Network Time Protocol
OSIA	Out-of-order Scheduling In-order Arrival
PDU	Protocol Data Unit
PDV	Packet Delay Variation
PFI	Path Failure Impact
PF	Potentially Failed
PDR	Packet Drop Rate
PLR	Packet Loss Rate
POSIX	Portable Operating System Interface
PPS	Pulse per Second
PR	Partial Reliability
QUIC	Quick UDP Internet Connection
RED	Random Early Detection
RTCP	Real-Time Transport Control Protocol

RTO	Retransmission Timeout
RTP	Real-Time Transport Protocol
RTSP	Real-Time Streaming Protocol
RTTVAR	Round Trip Time Variation
RTT	Round Trip Time
RTX	retransmission
RWND	Advertised Receiver Window
SACK	Selective Acknowledgment
SCTP	Stream Control Transmission Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMSS	Sender Maximum Segment Size
SRTT	Smoothed Round Trip Time
SS7	Signaling System 7
SSH	Secure Shell
SSN	Stream Sequence Number
TCP	Transmission Control Protocol
TO	Transmission Overhead
TSN	Transmission Sequence Number
TTD	Time To Delivery
TTL	Time to Live
UDP	User Datagram Protocol
VM	Virtual Machine
VRRP	Virtual Router Redundancy Protocol
WebRTC	Web Real-Time Communication
ssthresh	slow start threshold
WiFi	Wireless Fidelity

List of Figures

1 Schematic of the system's environment 2
2 Protocol Layering of the Internet Architecture Model 8
3 Components in a Streaming Application 18
4 Determination of the effects of absolute delay by the E-model. (Source: [46]) 24
5 End-Site Multihoming 26
6 End-Host Multihoming 26
7 Congestion Window Trace [64] 33
8 Transmission-Induced Sender Buffer Blocking 39
9 Advertised-Window-Induced Receiver Buffer Blocking 39
10 Reordering-Induced Sender Buffer Blocking 40
11 Measurement of Time To Delivery 44
12 Measurement of Transmission Overhead 45
13 Packet duplication through Fast Retransmission 47
14 Fast Retransmission not working anymore 47
15 Full Packet Duplication 50
16 Full Packet Duplication on link failure 50
17 Deadline Based Preemptive Retransmission 51
18 Deadline Based Preemptive Retransmission on link failure 51
19 Deadline Based Preemptive Retransmission on slow path 52
20 Basic Elements in a GStreamer Pipeline 54
21 The test setup in GStreamer 55
22 The test setup in GStreamer 58
23 The test setup with the measurement points 68
24 The test execution and evaluation script setup. 69
25 TTD Trace Parameters: `dpr`, $\rho = 12.5\%$, $\delta_e = 80ms$ 73
26 DDR over packet drop rate ρ for different setups Parameters: $\delta_e = 20ms$ 74
27 DDR over packet drop rate ρ for different setups Parameters: $\delta_e = 40ms$ 74

28 DDR over packet drop rate ρ for different setups Parameters: $\delta_e = 80ms$ 76
29 DDR over link delay δ_e for different setups Parameters: $\rho = 0.6\%$ 76
30 DDR over link delay δ_e for different setups Parameters: $\rho = 6.2\%$ 76
31 DDR over link delay δ_e for different setups Parameters: $\rho = 12.5\%$ . . . 76
32 TTD CDF for different packet drop rates ρ Parameters: `single`, $\delta_e = 20ms$ 77
33 TTD CDF for different setups Parameters: $\rho = 12.5\%$, $\delta_e = 80ms$ 77
34 TTD CDF for different DPR Parameters: `dpr`, $\delta_e = 80ms$ 77
35 TTD CDF for different link delays δ_e Parameters: `dpr`, $\rho = 12.5\%$ 77
36 DDR over packet drop rate ρ for different setups Parameters: $\delta_e = 20ms$ 79
37 DDR over packet drop rate ρ for different setups Parameters: $\delta_e = 40ms$ 79
38 DDR over packet drop rate ρ for different setups Parameters: $\delta_e = 80ms$ 79
39 DDR over link delay δ_e for different setups Parameters: $\rho = 0.6\%$ 79
40 DDR over link delay δ_e for different setups Parameters: $\rho = 6.2\%$ 80
41 DDR over link delay δ_e for different setups Parameters: $\rho = 12.5\%$ . . . 80
42 TTD CDF for different setups Parameters: $\rho = 0.6\%$, $\delta_e = 20ms$ 81
43 TTD CDF for different setups Parameters: $\rho = 12.5\%$, $\delta_e = 80ms$ 81
44 TTD CDF for different link delays δ_e Parameters: `dpr`, $\rho = 6.2\%$ 82
45 TTD CDF for different link delays δ_e Parameters: `dpr`, $\rho = 12.5\%$ 82
46 TTD CDF for different packet drop rates ρ Parameters: `dpr`, $\delta_e = 40ms$ 82
47 TTD CDF for different packet drop rates ρ Parameters: `dpr`, $\delta_e = 60ms$ 82
48 TO over link delay δ_e for different setups Parameters: $\rho = 0\%$ 83
49 TO over link delay δ_e for different setups Parameters: $\rho = 12.5\%$ 83
50 TTD Trace with path failure Parameters: `cmt`, $\delta_e = 60ms$ 85
51 TTD Trace with path failure Parameters: `udpdupl`, $\delta_e = 20ms$ 85
52 TTD Trace with path failure Parameters: `dpr`, $\delta_e = 20ms$ 85
53 TTD CDF for different setups Parameters: $\delta_e = 60ms$ 86
54 PFI over link delay δ_e for different setups 86

List of Tables

1 Notations 8
2 Fault Model 25
3 Comparison of Transport Layer Protocol Features (Inspired by: [80]) . . 37
4 SCTP Socket Options 61
5 Notations 62
6 Expected DDR Results 63
7 Packet and Header Length 64
8 Parameters for DDR measurement setup 72
9 Parameters of the video source 72
10 Parameters for PFI measurement setup 84
11 Measured Results With Modified Congestion Control 108
12 Measured Results With Regular Congestion Control 114
13 PFI measurement results 115