FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Using XMPP for System Monitoring and Administration

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Mag.rer.soc.oec.

im Rahmen des Studiums

Informatikmanagement

eingereicht von

Adi Kriegisch
Matrikelnummer 9625495

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Dr. Werner Purgathofer

Wien, 15.11.2015

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Using XMPP for System Monitoring and Administration

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Mag.rer.soc.oec.

in

Informatics Management

by

Adi Kriegisch
Registration Number 9625495

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.-Prof. Dipl.-Ing. Dr. Werner Purgathofer

Vienna, 15.11.2015

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Adi Kriegisch
Leystrasse 23/11/27, 1200 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Abstract

The purpose of this work is to muster many of the useful tools for managing and monitoring an infrastructure by using XMPP as middleware, using its advanced features and relying on its builtin security features.

By focusing on small to medium sized networks, that very often do not use sophisticated management and monitoring solutions, but rely on SMTP for the monitoring part and do management most of the time either manually or by scripting certain reoccuring tasks.

A common subset of tools will be selected and adapted for use with XMPP as its communication protocol. Wherever possible, information will be aggregated, stored and evaluated in an automated way. This ensures, that enough information is available for troubleshooting, while not overloading a systems administrator with status messages.

Several aspects of system monitoring and management will be dealt with, including event creation and performance data collection as well as creating the possibility to run special distributed tasks easily. The latter especially requires an easy to use interface that allows systems administration to add and integrate already existing special purpose scripts.

This tool chain is not meant to be a complete framework that just does “the right thing”, but is rather an effort to establish safe and secure best practices for communication middlewares useable for system management and monitoring. It contains lots of ready to use recipes and may be used for adaption and expansion.

Kurzfassung

Diese Arbeit versucht, aus den vielen nützlichen Werkzeugen, um eine IT Infrastruktur zu verwalten und zu überwachen, einige auszuwählen und zusammen mit XMPP, mit seinen vielfältigen Protokollerweiterungen und seinen ausgeprägten Sicherheitsmechanismen, als Nachrichtensystem zu verbinden.

Der Fokus für ein derartiges System liegt auf kleineren und mittleren Netzwerken, die sehr oft keinerlei Überwachungs- und Verwaltungslösungen einsetzen. Statt dessen wird SMTP für Benachrichtungen über den Systemzustand verwenden und Verwaltungsaufgaben werden manuell oder mit einfachen Skripten für wiederkehrende Aufgaben erledigt.

Eine Auswahl an Werkzeugen zur Systemüberwachung wird getroffen und für die Verwendung mit XMPP adaptiert. Wo immer es sinnvoll ist, werden anfallende Informationen automatisch aggregiert, ausgewertet und gespeichert. Das stellt sicher, dass ausreichend Information zum Lösen von Problemen vorhanden ist und gleichzeitig die Systemadministration nicht mit einer Flut an nicht direkt verwendbaren Information überlastet wird.

Weiters werden einige Aspekte von Systemüberwachung und Verwaltung detailierter behandelt: das Aggregieren von Systemmeldungen zu Ereignissen, das Sammeln von Performancedaten und die Möglichkeit, Aufgaben zu verteilen und auszuführen. Letzteres erfordert, dass einfache Schnittstellen für Systemadministratoren existieren, um ihre Aufgaben innerhalb des Systems abzubilden.

Das entstandene Instrumentarium versteht sich selbst nicht als eine vollständige Lösung, die einfach “das Richtige” tut, sondern eher als Versuch, eine Musterlösung für den sicheren und zuverlässigen Einsatz von XMPP als Nachrichtensystem für die Systemverwaltung und Überwachung zu etablieren. Weiters wird eine Reihe nützlicher Rezepte für die Verwendung und Erweiterung zur Verfügung gestellt.

Contents

"It takes about the same amount of computing to answer one Google Search query as all the computing done – in flight and on the ground – for the entire Apollo program."

Udi Manber and Peter Norvig, Google

CHAPTER 1

Introduction

Today's computing offers many challenges [102] and no matter how they get solved, a massive and ever growing computational effort is necessary when facing those. Years ago, one UNIX™machine provided most of the resources a department in a mid-sized company needed while today, a complex network of many different servers – most of them in a virtualized environment – provides a plethora of services to the same amount of users. This offers some challenges for those in charge of providing the infrastructure.

A wide variety of specialized tools exist that aim at assisting to fulfill parts of those tasks. Most of these tools bring their own messaging mechanisms making them hard to extend and reuse in other tools. They also implement their own security model that in most cases isn't well audited and brings its own authentication system.

Information about the state provides deep insight about inner workings of a network and may help attackers to select an attack vector. This kind of information disclosure should therefore be avoided. Furthermore some protocols used in testing a host's status even allow remote code execution and may be abused to take over a host system.

Central management systems accomplish their purpose by having access to all systems on a network and by executing commands on those systems. Proper authentication, account management and trust relationship is a requirement in this domain.

Bigger organizations can provide a network infrastructure that separates management and monitoring networks completely to mitigate security issues by just avoiding exposition. The security issues remain.

Most of the tools available implement their own network protocols and authentication systems. The more tools are used to manage and monitor a network, the more different protocols need to get secured and the more different account systems need maintainance. XMPP (eXtensible Messaging and Presence Protocol) is a push based protocol that provides advanced features like adhoc commands, enhanced status and presence reporting, publication, aggregation and subscription of messages. This protocol also allows federation which may be used to share information with others and provides strong authentication and security features.

The purpose of this thesis is to bring together many of the useful tools for managing and monitoring infrastructure with XMPP as middleware using its advanced features and relying on its builtin security mechanisms.

A common subset of tools and protocols will be selected and adapted for use with XMPP as its communication protocol. Whereever possible, information will be aggregated, stored and evaluated in an automated way. This ensures that enough information is available for troubleshooting while not overloading a systems administrator with useless or redundant information.

Several aspects of system management will be dealt with, including enforcing common setup and configuration and creating the possibility to run site-specific distributed tasks easily. The latter requires an easy to use interface that allows systems administration to add and integrate already existing scripts written for a specific purpose.

This tool chain is not meant to be a complete framework that just does “the right thing”, but rather is an effort to establish safe and secure best practices for communication middlewares useable for system management and monitoring containing lots of ready to use recipes and being ready for adaption and expansion for small and mid-sized networks.

"our perceptions of being 'in control' always have a lot to do with scale as we focus our attention – and, by implication, the information that is omitted. We sometimes think we are in control because we either don't have or choose not to see the full picture"

Mark Burgess, In Search of Certainty: The Science of Our Information Infrastructure

CHAPTER 2

Overview

Operating a small to medium sized network is treated as a side job running in the background by most – their work runs on top of a working infrastructure treating everything below as a given foundation, something existing. Providing such an infrastructure requires effort, providing such an infrastructure in a safe and stable way means a lot of work. Unfortunately, most of the time there just aren't enough resources particularly with regard to time, man power, knowledge or money available to make the infrastructure available and lots of companies and organizations end up working on top of an infrastructure barely able to protect their valuable work while still requiring resources for basic operation.

Network operation has been defined by the International Standards Organization (ISO) in ISO 10040 (ISO/IEC 7498-4 [123]) to consist of five tasks – also known as **FCAPS** – that fit very well for systems administration as a whole:

- **Fault** management – detect and correct faults in a network and prevent the same or similar faults from happening again. This also includes documentation and recording of events and error logs in order to be able to make learning how a fault could have happened in the first place possible.

  Providing documentation on how to react in case of a fault to the operating personel by giving them handling instructions is also part of fault management.

- **Configuration** management – keep track of configurations of all devices and systems and track their changes. This aids in diagnosing and troubleshooting as many issues arise from configuration changes. In the ISO definition 'configuration' also includes software updates. Having configuration data and installed software available also helps in provisioning new systems, planing and extending.

- **Accounting** management – gathering usage data of systems and services for resource planing, redistribution of resources and possibly billing for resource usage. Billing is either relevant for bigger organizations where there is an IT department that sells their

services or where rental of computing resources is part of the core business like an Internet Service Provider (ISP) or a compute cluster or storage systems with a pay per use model.

- **Performance** management – constant monitoring of resource usage to ensure that all systems operate at their full speed and to be able to identify possible problems early, either by trending usage (long-term, rather related to accounting management) or by detecting overload or reliability issues (short-term, related to fault management from above).

- **Security** management – applying, enforcing and monitoring of security policies which includes the collection, examination and distribution of security relevant information and events. As this is an important topic on its own, there exists an accomodating document solely dealing with aspects of security management in networked environments known as ISO/IEC 7498-2 [122]. In essence, security management is a process that ensures **Confidentiality**, **Integrity** and **Availability**.

Probably one of the most important aspects defined in ISO/IEC 7498-4 and 7498-2 is the definition of security management as an ongoing process. Even today, more than 25 years after the release of that standard, security and security management is regarded as an externality and something to be dealt with by buying "a box of security" consisting of anti virus software, a firewall, a web-washing proxy or something else with the sole purpose of supplying more security without the need to act or put any more resources into maintaining the state of being secure. Buying the box is just a starting point for security management, just like an anti virus software needs constant updates, sometimes upgrades and at least a little attention.

2.1 Security Aspects

From Confidentiality, Integrity and Availability – in short the "CIA triad" – two very important aspects – namely authentication and non-repudiation – are missing to complete the list of computer security attributes, abbreviated as "CIANA". The following sections will refer to these attributes, therefor they shall be detailed here.

Confidentiality

Information is to be kept inaccessible outside the group of authorized viewers, it must not leak or be disclosed. Groups consisting of people or machines in case of automated information sharing need to be defined by policy.

From a technical point of view, confidentiality requires some kind of access control – be it file system permissions on a file system, encryption techniques while data is in transit on an untrusted network – which in turn requires some kind of authentication mechanism to verify the identity of the person or machine requesting access to the information.

Integrity

Integrity means that information in question is unmodified, complete, correct within its scope and has not been tampered with. The "Federal Standard 1037C" [181] defines integrity on several levels that are all relevant to system management and monitoring:

- **data integrity** – data remains unchanged and complete, even through transfer or storage and fulfills expectations about its quality throughout its life time. Reasons for modifications – technical or malicious – do not matter in that context as long as they can be detected.
- **service integrity** – the correctness and unimpairedness a service fulfills its purpose with.
- **system integrity** – an automated information system is unmodified, unimpaired and operates within its limits.

Availability

Information and systems providing information need to be available. The danger of unavailability lays in either a lack of information and services or an incomplete view on data that may result in wrong decisions. As well as authentication with an unavailable authentication service will not work, finding a fault in a complex network without access to log information will be more difficult.

Authentication

Means of verifying the identity of an actor, be it a human or a machine, for the purpose of deciding whether to give access to resources or information or not. Furthermore the ability to trace access back to actors aids in avoiding repudiation issues as even modifcations by authorized actors can be malicious in nature.

There are basically two authentication models: mutual authentication which fits for entities that already "know" each other or exchanged secrets and trusted third party authentication like a Kerberos service in a network, an OpenID provider or in real life the issuer of a passport or a different document suitable for authentication.

Non-repudiation

Non-repudiation deals with the ability to trace information and operations: certainty about the origin of data or the access to a service.

Providing security always means taking every single aspect of information security into account; only dealing with a few of them helps in gaining the feeling of being secure at best but leaves the door wide open for malicious activity or failures. Another common problem are assumptions: software or protocols designed with a closed down network in mind should not be deployed over the internet in the very same way; the absence of basic parameters prohibits that.

2.2 Security Requirements in Monitoring and Management

Monitoring as well as Management systems deal with potentially confidential information: states and transactions within a computer network. Access to that kind of information is very helpful for an attacker or a competitor trying to gain insights on a network. Monitoring systems provide details about available resources, impact of certain requests to applications, tools used internally and more (see Section 3) whereas management tools allow gaining insights on workflows, routine tasks, possibly passwords and more. Both of which is better not disclosed.

Trust relationships between entities on a network are mandatory for not disclosing information. Furthermore, every tool that triggers a command execution, be it a monitoring tool testing for some resources or services or a management tool trying to adapt a configuration, needs to prove its identity to the entity it wants to run a command on. Violating that requirement may either lead to denial of service (DoS) attacks on the machines by running a command too often, abuse of available commands for other purposes or even arbitrary command execution. For a closed monitoring-only network, access control and identification based on IP addresses may be sufficient, provided all rogue or hacked machines with access to that network are being detected and removed immediately. In open access networks, a third party like an LDAP or Kerberos server or a certificate authority may provide an authentication service for machine to machine communication in a network.

With confidentiality and authentication in place, integrity is another essential feature: all communication must be transmitted complete and unmodified. The impact of violating this requirement on a monitoring system leads to erroneus and incomplete data. This may happen either due to technical issues or by malicious activity. Basing decisions on either incomplete or modified information is impossible. The operators of GitHub therefor developed Brubeck [8] (for details see the analysis of statsd in section 3.5) because the statistics collector they were using before, lost metrics and their data sets were incomplete and therefor unusable. In case of system management, the inpact of incomplete or modified commands may lead to even more serious problems.

Information not only needs to be complete and unmodified but it also needs to be available when the operator needs it as a foundation for decisions: Lost messages about a failing hard disk give a wrong idea about the state of a network just as does the lack of list of installed software packets on servers when considering a security update – especially in the light of an ever shrinking time window between the public availability of a patch and the disposability of mass infection tools exploiting the very bug [103].

All information in a network has an origin and with the help of a trusted authentication system, it is possible to verify the source. This leads to non-repudiation which helps in two ways: First, together with authentication it makes it impossible to inject wrong or misleading information into a monitoring system and second, the origin of malicious activity with a wrong identity, credentials stolen from a user or a machine can be tracked.

Monitoring Protocols

General aspects of security as discussed before were already applied to monitoring protocols: Adrian Perrig [159] named *Data confidentiality*, *Data authentication*, *Data integrity* and *Data*

freshness as design criteria for the secure sensor network protocol he developed. The protocol is designed to run on very limited hardware that is unable to handle a decent crypto stack. “Data freshness” in the context of that protocol can be seen as availiability and non-repudiation is gained through the data authentication.

These criteria are in line with what Kenneth E. Nawayn [150] named as where the syslog protocol is lacking. In section 3.3 a more detailed analysis of the protocol, its shortcomings and its current use will be presented.

2.3 Fault Detection and Handling

Collecting availability, operation status and performance data of systems is a requirement for providing reliable IT services. Just relying on availability checks has the disadvantage that a reaction to an incident is only possible after it happened (reactive). The occurance of many issues is however predictable: Hardware issues like a failing harddisk may be recognized before the disk actually stops working by observing their built-in health data (proactive), a partition filling up is detectable in most cases too (predictive). As well as environmental sensors like ambient temperature give good indication of something going wrong, a permanently loaded CPU does so too.

An early reaction to the aforementioned events can avoid an outage or at least help lowering its impact. Furthermore, the data is helpful in future resource planning.

The purpose of monitoring is to assist operational personnel to be able to

- **act** before an outage, either by being able to predict something is going to happen and/or by proactively dealing with an event that is about to take place. If a problem remains undetected until it actually happens, the operators should at least be able to recognize the issue on their own and **react**.

- **plan** resources necessary to continue trouble-free operations. This includes having an overview on the development of resource usage over time. Having that kind of information at hand it is possible to make better use of available resources and to decide when to replace or extend currently used systems.

- **troubleshoot** and debug errors happening on systems or services. Detailed performance data and corresponding events help at finding faulty components even in a more complex tool stack. Debugging is an interactive task that requires an experienced operator and cannot be automated.

The goal is to know the states of a network, servers and services. Analysis of events may result in state changes, eg. from a known good or operational state to a failed state or even an unknown state, that requires actions to change the state back to an operational state again. Some of these actions may even be automatic actions triggered by either the monitoring system or one of the systems involved in the state change. Cluster and High Availibility software are typical representatives of the latter category. In recent years many software systems offer the ability to guard services like for example supervisord [60], uwsgi [63] or even systemd [62] – a

Linux startup system that starts all necessary processes like the original sysvinit and supervises these processes and reacts on crashes. Automatic reaction requires a reliable detection of the root cause of the state change. For more complex state changes that involve interdependencies, automatic remediation is very difficult and may lead to more problems when done wrong. To be able to act, sufficiently complete monitoring has to be done, so that decisions can be based on data about a network as a whole.

Data Acquisition from Legacy Systems

There are, however, plenty of systems like switches, printers or storage systems in a network that are closed and cannot be monitored from the inside. Most of them provide at least some standardized mechanisms to gain insights on their current operational status like SNMP (see Section 3.4 for a detailed description of the protocol), Syslog (see the evalutation of syslog in 3.3) or some kind of web interface. Collection and aggregation of this data may prove to be difficult but offers advantages over just observing from the outside and treating these systems as a blackbox:

- Taking an overloaded switch as an example, the behavior that may be observed from the outside are random connection drops, stalling connections, the impression on user side that the DNS service isn't working anymore or the inability to recieve an IP address via DHCP [100].

  For the operator there are plenty of places to look for the error, beginning with loose network cables, the firewall, the name service, the DHCP server, the user's desktop machine and so on.

  Observing the dropped packet count in the switch or its CPU usage via SNMP or error messages sent out via syslog, the problem might have been detected earlier and in a more reliable and stress-free way.

- Another example may be a networked printer in a dedicated room having either a paper jam or running out of paper or toner. These issues will be detected by users when they try to pick up their print job. Most of the time, their reaction will not be helpful either, because they will try to get their job printed. They don't care for the printer or all print jobs in the queue; their interest is their own print job. Most probably they are under time pressure and experience this kind of trouble as an unnecessary interrupt to their work.

  Most printers offer several possibilities to gain operational data, event logging via either Syslog or SNMP, alerting via SMTP and general statistical data via SNMP.

  Doing blackbox monitoring on printers leaves two opportunities: checking whether the printer can be reached (and it will be reachable during a paper jam too) and queue time monitoring which makes it possible to gain some insights on things going wrong but will also have a high false positive rate due to for example print jobs that will take longer to complete due to complex graphics or their sheer length.

Data Classification

Data that can be acted upon is clearly an event like “hard disk broken”, “service crashed” or “paper jam”. But there is statistical data coming from an average system like CPU performance data, disk or network usage and so on. As stated above, this data should be collected to aid in debugging or resource planing but may also be used to create events like “disk 95% filled”, “CPU overloaded” and so on.

All this data must be enriched with prior knowledge on network, systems and services, MAC and IP addresses as well as device interconnects, service dependencies, available sensors, geographic locations and dependencies between sensors, devices and event sources. Only with this kind of knowledge accurate diagnosis and efficient reaction to incidents are possible.

Event Chronology

Events trigger actions – automatic or manual – that influence the state of a network, system or service. Integrity and completeness of performance and event data is necessary as is timeliness and synchronicity. A minimum requirement is to synchronize clocks on all systems monitored, because data without an associated accurate time stamp cannot be correlated with data aquired from other systems in a network. Also reacting on long past events will lead to confusion and destabilize a network.

2.4 Network Organization

Information about the state of a network needs to be collected, analyzed and reacted upon. This can be done in a centralized way, having for example a system with the sole purpose of aggregating information about the state of services and devices or in a rather autonomous way, letting the device supervise itself, evaluate the collected information and act or notify about what is going on.

A centralized approach to monitoring and management has long been the standard and partly still is for several reasons: many of the devices in a network providing a service were limited by the available resources or the ability to run custom code that would allow them to react on certain events in a meaningful way. Furthermore it was uncertain wether a failing system would be able to detect its state and react to the events leading to that state in a meaningful way.

With a rising complexity and more powerful computing possibilities, decentralizing monitoring and management became feasible [111] and even more so a necessity for reducing the overhead of collecting and aggregating metrics for a whole network [99]. Detecting failures from within a failing system in a reliable way remains a challenge that still hard to completely solve as there exist error conditions that cannot be detected from within the usual monitoring means provided. It, however, turned out that self-monitoring faces the same challenges as centralized monitoring when it comes to observing the blind spots like certain hardware failure conditions.

Promise Theory

One of the proponents of decentralized structures in system management and monitoring, Mark Burgess, formalized this approach in his more universal promise theory [76]. At the core, this approach models interdependency of systems through promises – offers to fulfill certain tasks and provide certain services – in contrast to obligations to do the very same thing as with a centralized or contract driven models. Promise – as a word – includes the possibility of failure which leads to a model that better fits the reality of networks and systems.

The centralized, obligation based approach – also called the "Command and Control" approach – separates obligation from implementation which leads to uncertainty about the outcome: instructions to act are issued at a central place that first needs to gather relevant information about current state of the system and surrounding conditions which is already present on the system itself. Furthermore, a command based system may lead to issues with contradicting commands that, in the non-independent obligation based approach cannot be resolved on the local system.

The promise based approach allows to have expectations based on the promises of services and helps the engineer to combine those services and components to a complete system fulfilling requirements for a new or more complex service. Mark Burgess described this in a 2014 article in Linux Journal [75]: "Electronics are built in this way, as is plumbing and other commoditized construction methods. You buy components (from a suitable supplier) that promise certain properties (resistance, capacitance, voltage-current relationships), and you combine them based on those expectations into a circuit that keeps a greater promise (like being a radio transmitter or a computer)."

Modelling systems and services following this approach results in a dependency graph allowing a deeper understanding of dependencies and weak links. Failures and their consequences are embraced in the model itself.

2.5 A System Administrator's View

Going back to the beginning of this chapter, operating a small to mid-sized network is, in most cases, a side job either done by someone working at the core business of the organization owning the network as main profession or by someone operating other networks of similar sizes too. Both cases have in common that the time and resources used to operate the network are shared with other tasks.

The daily business of a system administrator is centered around a few tasks:

- keeping all servers and services running mostly by ensuring they are operating as they should
- patch and update servers and services
- react on external information about security threats or potential problems
- execute complex routine tasks that involve different actions on different machines like the creation of user accounts

- collect information about what is going on in the network
- correlate events like a high reject rate on a mail server with either a new spam wave or a malfunction of a component necessary to accept mails

These tasks sum up to an ongoing effort in system administration and may, due to their repetitive nature, be partly automated.

Services and Management

Central structures with the sole purpose of aggregating information about the state of services and devices are often rightfully regarded as overkill. Tasks like analyzing log and error messages, evaluating service state, sending notifications, possibly reacting to events and providing a user facing service can easily be done on the devices themselves.

When deploying new services, system administrators very often follow a promise based approach even though they almost never explicitly thought about it that way: They pick components able to fulfill parts of the requirements for a new service. Keeping the dependency graph resulting from the initial deployment already provides a big part of network documentation and may also be used to gain a deeper understanding of what is going on in a network in case of a failure.

Using a promise based approach to system design and network also aids in knowledge sharing and documentation. Delegation of parts of the administrator's work is easier, on upgrading and replacing single components all the promises other parts depend on are obvious and to recreate a similar service at another site a list of requirements and dependencies is available.

"In a data deluge–era sensing system, the number and resolution of the sensors grow to the point that the performance bottleneck moves to the sensor data processing, communication, or storage subsystem."

Dr. Richard G. Baraniuk, 'Science Magazine' 02/2011

CHAPTER 3

System Monitoring

A plethora of tools dealing with certain aspects of monitoring or system management exist. In the course of this chapter some of these will be presented and evaluated. The most important aspects of the tools evaluated are network data exchange and the messaging subsystem in general. Starting with general evaluation and classification criteria the chapter progresses to monitoring tools and closes with a summary.

3.1 Criteria and General Considerations

The main focus of the following chapter is on the message distribution and network communication aspect of a chosen tool. Besides general security considerations as detailed in 2.1, the kind of data exchanged needs to be analyzed too.

Comparing Monitoring Tools

Monitoring tools collect, aggregate, store and evaluate huge amounts of data. While every tool focuses on certain aspects ranging from graphical representation, automated alerting and data analysis to the collection of local performance data or events, using certain protocols like SNMP for data aquisition or the organization of storage for performance data, there is a common set of functionality they provide.

Joshua Barratt proposed the following categories of functionality a monitoring tool provides [72]:

- **Collect** – tools that spefically deal with making certain kind of sensor data available for further processing.
- **Transport** – tools that provide a network transport for exchanging monitoring data.
- **Process** – tools that inspect and analyze data to gain further insights into the data or to raise alarms on possible errors.

- **Store** – tools that store data either for archiving purposes or for other tools to use them.
- **Present** – tools that allow interaction with an operator by presenting the available data. In most cases, these tools provide web interfaces that allow interaction with the available data.

Borders between these categories seem, of course, fluid: a tool that is able to read a temperature sensor needs to do some processing of the data in order to be able to present a value to the user. Assuming that said tool is a command line utility that prints out a temperature after being called, this tool only fits in the "Collect" category of the above, maybe fits in the "Process" category when it also does some alerting but definitely does not fit in the "Present" category.

Exchanging data [184] between different monitoring tools is more and more important, even experiments with unified data schemes [152] have been tried. The more of the above aspects a tool tries to cover the harder it usually is to integrate it with other tools or to replace parts of its functionality. Tools that empower the classic UNIX™concept of "do one thing and do it well" are, most of the time easier to integrate in a tool chain. Unified data schemes tend to appear over-engineered and are overly complex to use; over the course of the past 20 years, tool stacks that glue together many smaller tools dominated the field of system monitoring.

Time and Data Correlation

Using a common time source is vital for the usefulness of data acquired as this is the datum that allows correlation of events and statistics acquired. As long as monitoring involves only a single host, synchronized time is of no importance, because all events are being logged and collected based on the same relative time but as soon as events and statistics from multiple devices are being collected, their system time needs to be synchronized and more, in many cases, it is even necessary to take time zones into account, either by using Coordinated Universal Time (UTC) or by making sure, that the whole software stack is able to deal with time zones in an appropriate way.

3.2 Time Series Databases

Most storage subsystems for monitoring tools are based around time series databases, automatic expiry of data and their aggregation. Besides the classic syslog text with its logrotate that rotates and compresses older log files and removes them after a certain period of time, several time series databases are available that are designed specifically for sensor data. The use of a classic relational database management system (RDMS) for this purpose is, of course, possible but – due to the nature of data stored – not desired as better fitting solutions exist.

Time series are a special kind of information that are often dealt with in system monitoring: at its core a time series consists of a sequence of measurements taken at a certain interval. Most of the data needs normalization like the transformation into a rate. In the easiest and most obvious case, the amount of packets sent and recieved by a router during a certain period of time forms a rate by taking the absolute number of packets sent and recieved between two distinct points in time and dividing them by the amount of time elapsed. Other data, like the temperature

in a server room, does not need to be normalized as it is not aggregable and average CPU usage over a certain period of time is a value that has been normalized before aquisition.

Round Robin Database – RRD

To date the most popular time-series based database is RRD (Round Robin Database) [154] developed by Tobi Oetiker. It stores its data as files in the file system in a way that allows the files to be copied over to another system and be used there; a property that proves to be useful when separating data aquisition from graphing and analysis. In contrary to regular SQL based database systems, all data is arranged around time following a storage scheme known in advance: at database creation time, the interval at which new data points arrive and the number of data points to store needs to be known. Data retention times can be derived from those values. After creation, the database file never changes its size because it gets created with all data points containing empty values. This offers two advantages: First, the file is created around the same place on the physical drive, which is a huge performance advantage, because the database isn't fragmented on the media, at least on rotating disks, and second, as the space used by the database is known and used in advance, even a full disk will still allow data collection as already existing data simply gets overwritten.

Data points, before being added to a time series database, may need transformation. An example for such a conversion is the transformation of counter values into a rate. After that, data needs to be normalized within the time range

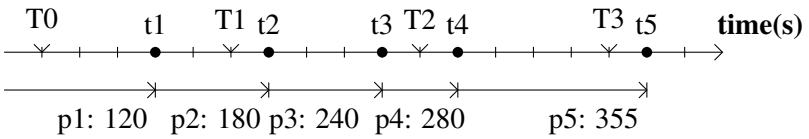

Figure 3.1: time series data

Data Normalization

Figure 3.1 shows a timeline with fixed interval borders (T0-T3) and data points (p1-p4) arriving at certain points in time (t1-t4). RRD collects these Primary Data Points (PDP) and aggregates them into Consolidated Data Points (CDP). In the first step PDPs are normalized. This happens according to their specification: RRD offers several distinct data types that all are normalized in a different way:

- `COUNTER` – in network monitoring one of the most common data types as found in most switches and routers. The counter itself can only increase, the normalization function subtracts the former value from the current one to get the delta or rate for the time between the the current and the last data point.

  Even though counters are only ever increasing, there is still network hardware in wide use that only provides 32bit counters for the amount of transmitted data. With current network

speeds (gigabit and above), those counters can overflow every 32 seconds. RRD assumes a counter overflow when a counter decreases and adds the difference to the normalized value.

- `ABSOLUTE` – specifies a counter that gets reset every time it is read, so there is no need for a subtraction to get the current value as with counter. Aside from that it is treated the same as a counter: the value is asumed to be the rate for the time between the current and the last data point.

- `DERIVE` – is also a counter like type for which the change rate compared to the last datum is calculated. Therefor the result of this calculation may yield negative numbers too, indicating a decline.

- `GAUGE` – for this data type, no calculations are performed, as an absolute value being constant (or already averaged) for the interval between the current and the last data point is assumed. Example for data that fit into this category are temperature or the 5 minute load average, the UNIX™ `uptime` command provides.

In this very first step, data is converted into a rate: the change of a sensor value over time. The second step of data normalization, the consolidation, deals with the alignment of the data within the defined interval: Data points may not arrive at the exact time they are expected to (as can be seen in the example in figure 3.1), some data sources – like, for example, switches with 32bit counters – require the data to be pulled more often than the interval given for data collection or values which are too volatile to be measured just once at the given interval. The consolidation process results in CDPs that finally get stored in the database.

Data Consolidation

RRD offers four distinct consolidation functions that result in CDPs:

- `AVERAGE` – calculates a weighted average of rates for the interval. Data collected that way has some very convenient properties that can be used in billing for example: the counters of routers being consolidated with this function allow to multiply the rate by the time to get the absolute number of bytes sent and recieved.

- `MIN` – stores the smallest value recieved during an interval. Useful for collecting data about thresholds and for trending.

- `MAX` – is similar to `MIN` but stores the biggest value recieved.

- `LAST` – stores the last rate recieved in an interval.

`MIN`, `MAX` and `LAST` modify the original data in a way that a certain kind of information gets lost and the data has different properties than what was originally measured. As mentioned above, when using `AVERAGE`, most of the original information is preserved, just details about highs and lows are removed. The data, however, is still countable [183].

Normalization and Consolidation by Example

Going back to the example shown in figure 3.1, the normalization and consolidation of data would look like this. To make calulations easier, it is assumed that p0 is 0 and arrived just a time tick before T0. The data type is `COUNTER` and the function used for consolidation is `AVERAGE`.

PDP	raw	normalization	value
p0	0		0
p1	120	120 - 0	120
p2	180	180 - 120	60
p3	240	240 - 180	60
p4	280	280 - 240	40
p5	355	355 - 280	75

Table 3.1: normalization of primary data points

The normalization process in table 3.2 shows simple counter normalization without any overflow happening: all values read from the counter are increasing. A value of 0 for p0 could be an indicator of either a restart of the system monitored or of a counter overflow.

interval	PDPs	weight	normalization	CDP
I0: .. - T0	p0			(0)
	p1	0.25	120 * 0.25	30
I1: T0 - T1	p1	0.75	120 * 0.75	(90)
	p2	0.66	60 * 0.66	130
I2: T1 - T2	p2	0.33	60 * 0.33	(20)
	p3	1.00	60 * 1.00	(80)
	p4	0.50	40 * 0.50	100
I3: T2 - T3	p4	0.50	40 * 0.50	(20)
	p5	0.80	75 * 0.80	80
I4: T3 - ..	p5	0.20	75 * 0.20	(15)
	..			

Table 3.2: consolidation of primary data points

After normalization, consolidation takes place as shown in table 3.2. Every normalized data point represents a rate showing the activity within the time span between the the current and the last data point. Collection interval (I0-I4) are also between well defined borders and the data points get aligned to those fixed collection intervals. Note, that the sum of all CDPs (including the incomplete CDP for I4, of course) is the same as the sum all normalized PDPs and, of course, the same as the raw counter value of 355.

Database Creation

In the following example, we create a RRD database that stores 1 hour of data for two counters – "in" and "out" – at a resolution of 5 minutes (300 seconds, the default for RRD), 1 day of data at a 30 minutes interval and one week at an hourly interval. The data gets aggregated by the "average" function:

```
rrdtool create in_out.rrd --step 300 \
         DS:in:COUNTER:600:U:U\
         DS:out:COUNTER:600:U:U\
         RRA:AVERAGE:0.5:1:12 \
         RRA:AVERAGE:0.5:6:48 \
         RRA:AVERAGE:0.5:12:168
```

The specification of the data sources (`DS:`) is followed by the specification of the different storage archives (`RRA:` or Round Robin Archive). The interval specification (`-step 300`) is used as the base for all further time and size calculations:

- 1 data point at 300 seconds is kept 12 times which means one hour of data at a resolution of 5 minutes.
- 6 data points at 300 seconds aggregated with `AVERAGE` result in one data point for 30 minutes stored 48 times which means one day at a resolution of 30 minutes.
- 12 data points at 300 seconds aggregated with `AVERAGE` result in one data point for one hour stored 168 times which sums up to one week of data at a resolution of 1 hour.

The archive in the above example takes about 5068 bytes of space on disk which consist of 1420 bytes of header and 8 byte of storage per data point, which corresponds to 64bit values.[1] Aggregation to lower resolution archives takes place at inserting new data.

Graphite's Whisper

Whisper [66] is the time series database of the Graphite project – a graphing system that renders graphs based on data stored by its backend on demand. As similar RRD and Whisper are in terms of storage layout and basic theory of operation as different both are in regard to their concepts. Neither of the two is a drop-in replacement of the other. Whisper only expects one data point for any given interval, it even offers the opportunity to insert older data points after newer ones were added. The database file itself not only holds the raw data but also stores the time stamp. Processing of the data, like conversion in a rate or averaging data can only be done in Graphite itself. Whisper expects exactly one data point per interval; in case more arrive, already existing data points will be overwritten.

Going back to the example time series in figure 3.1, Whisper would overwrite p2 with p3. To make the comparison of the results easier, normalized values of p0-p5 (see table 3.2 were fed into Whisper and the data points used to store were annotated. As it can be seen from table 3.2,

[1] these values are accurate for rrdtool version 1.4.7 and file format layout version "0003".

Interval	data points Whisper	data points RRD	Whisper	RRD
I0	p0	p0, p1	0	30
I1	p1	p1, p2	120	130
I2	p3	p2, p3, p4	60	100
I3	p4	p4, p5	40	80
I4	p5	p5	75	15
Sum			295	355

Table 3.3: Difference between Whisper and RRD

Whisper clearly expects the data to be collected and aggregated before being added to the storage backend. Graphite's toolstack provides Carbon, at networked daemon, that is able collect data via network but also does not provide a facility to aggregate data.

Provided that data is properly aggregated beforehand, Graphite allows interactive data exploration. For many this is a much desired feature that requires raw data and is therefore hard to do with data collected in RRD databases [126]. On the other hand, rates by themselves aren't precise information but already an aggregation of data: Knowing how many bytes were sent and recieved during the last five minutes does not give any idea about peak values or about wether these bytes have been sent during the first ten seconds or evenly distributed during the whole time frame. In other words, as Gregory Trubetskoy put it [179]: "Perhaps the biggest misconception about time series is that it is a series of data points. What time series represent is *continuous* rather than *descrete*(sic!), i.e. it's the line that connects the points that matters, not the specific points themselves, they are just samples at semi-random intervals that help define the line. And as we know, a line cannot be defined by a single point."

In contrary to Whisper, retrofitting of data in RRD is impossible because information about the interval a data point corresponds to is missing and so data points inserted into the database can only be younger with regard to the time stamp. The second border of the interval is always the the data point from before.

Whisper, on the other hand, has a completely different approach to the data normalization process: it is the data supplier's responsibility to normalize the data. At its core, Whisper only provides aggregation functions for moving data to lower resolution long term storage. In contrary to RRD it also stores the time stamp at which data arrived in the archive to allow retrofitting data sets and therefore also does not have the claim for correctness of its historic data. Besides storing data points it relys on the data provider to actually collect and preprocess the data to fit in a time series storage.

3.3 The Syslog Protocol

In system administration the term syslog is used to describe several different aspects of the syslog protocol that may only be distinguished through context:

- syslog as a **facility** which may be used by local processes to submit log messages to.

- syslog as a **network protocol** that is used to exchange and forward log messages between hosts.

- syslog as a **message format** that contains metadata and log information to be collected and stored.

The syslog protocol [138] itself has been developed by Eric Allman at the University of California at Berkeley for use with the first version of sendmail [55] in the early 1980's. The protocol soon became an unofficial standard used by many other tools for logging and was finally documented in RFC3164 [138].

At its core, the protocol uses connection-less UDP for network transport and requires the message to consist of three parts: PRI, HEADER and the original log message itself. PRI – the priority – consists of a number indicating the facility that originally generated that message and a number indicating the severity of an event, qualified by the sender. HEADER consists of a time stamp generated by the originating device and a host name; the sender's ip isn't used in the protocol except for network packet delivery because the protocol itself allows relaying of messages much like SMTP [161] [134] [135]. The message itself contains the component or application as its first string followed by a free text message.

By default syslog messages are stored in plain text files that are rotated by a tool named `logrotate` and kept for a definable period of time. Syslog daemons use priority, facility and application for filter definition that allows redirecting messages into different files. For example, storing all mail server related logs in a separate file like `/var/log/mail.log`.

As with many other protocols at that time, security wasn't a primary design goal for the protocol. Kenneth E. Nawayn [150] identified three major categories of problems where the syslog protocol is lacking:

- **Confidentiality** – network transmission of syslog data is in plain text.

- **Integrity** – syslog data may have been modified on the wire.

- **Authenticity** – the originator of a message cannot be verified.

Syslog messages can, due to their nature, be suppressed, modified or injected (spoofed) which very much limits the scope the protocol may be used in to small, trusted and isolated networks. Unfortunately, the syslog protocol is still used over the internet.

Use of Syslog over the Internet

A random sample of a part of the AcoNET backbone[2] from July 21st 2015 00:10h to July 24th 2015 11:40h shows pretty clearly that the original plain syslog protocol (port 514/udp) still is the predominant way of exchanging syslog messages, even over the internet.

While many lecacy systems only support the original syslog protocol, its use over the internet also shows a lack of attention by those responsible for managing these systems.

[2] Special thanks to Christian Panigl and Alexander Talos-Zens for providing these numbers.

Channel	Flows	Packets	Traffic	Protocol
514/udp	19700	52000k	12.2GB	plain syslog (RFC3164 [138]
6514/udp	156	471k	0.3GB	secure syslog (DTLS/RFC6012 [175])
6514/tcp	1200	1000k	0.6GB	secure syslog (TLS/RFC5425 [144])

Table 3.4: syslog traffic over the internet

Improvements to the Protocol

Several efforts to improve aspects of these shortcomings have been made by specifying protocol extensions like "Reliable Delivery for syslog" [151], "Transport Layer Security (TLS) Transport Mapping for Syslog" [144] or "Signed Syslog Messages" [133]. The main problems with these extensions are twofold: First, legacy devices do not support them so the least common denominator remains plain syslog as defined in RFC3164 [138] and second, making use of those extensions require at least building and operating an own PKI(see eg. [182]) or manual key distribution.

The first issue can be worked around with a directly attached syslog server that collects plain syslog messages and forwards them using above mentioned protocol enhancements while the second is a non-issue that just puts more workload and attention to the detail on the people in charge for administering a system.

Handling the Data Format

Automated processing of syslog messages brings some difficulties due to the nature of its data format. The oldest documented approach to this is probably Marcus J. Ranum's "Artificial Ignorance" [5] write-up from 1997 that proposes to drop known harmless event notifications by specifying regular expressions in order to be able to focus on what is left: messages that contain either known good (important in this context) events or unknown events. This idea is used in tools like Logcheck [34]. Although other approaches like "On the use of weighted syslog time series for anomaly detection" [104], "A wavelet-based framework for proactive detection of network misconfigurations" [139] or "Alert Detection in System Logs" [158] exist. The currently preferred way of dealing with syslog information is to feed it into Logstash [35], a general-purpose tool for collecting, processing and outputting log information. Together with Elasticsearch [21] – a full text search and analytics engine – it provides tools for searching and analyzing syslog messages.

Both of the above approaches – the statistical one and the search approach – have disadvantages: When dealing with syslog messages, one never knows what to look for, but what to not look for. Marcus J. Ranum's approach is therefore still the most reliable one, although the effort for creating all the required regular expressions initially is quite high, there already exist many of those for common services and tools like Logcheck [34] help a lot with getting started fast and as soon as the noise is removed, syslog information can be a viable source of information about events and states in a network. For many legacy systems they are the only source of information.

Improvements to Syslog

Rainer Gerhards, author of rsyslog [53], authored RFC5424 [109] that became standard in 2009 and brought many improvements to the original RFC3164 [138] that actually only described current implementations of syslog. A summary of the most important advances of the renewed message format that is meant to separate transport implementations like the one from RFC5425 [144] or RFC6012 [175] from the message format described in this RFC:

- A protocol version is included that allows future message format extensions without the need to "guess" the actual message format as it is the case with the slightly diverting implementations of the original standard.
- The timestamp format finally adds a year and a timezone, both of which weren't part of the original specification.
- Hostname, application name, process ID and message ID now have explicit fields which adds additional parsable information to a syslog message.
- The data format allows adding structured data in form of key-value pairs.
- The encoding of a message is by default UTF-8.

Of course the new format is not perfect in every aspect; criticism ranges from the chosen date format that is different from the ISO8601 standard to software vendors still not using the opportunity to use structured logging.

In 2007 Mitre started another initiative named "Common Event Expression" [36] (short: CEE) that lead to new, yet unfinished, standard formats for log messages based on XML and JSON data formats. Due to a lack of funding for that project, Mitre did not complete the project itself but is hoping for others to take over and continue the effort.

Project Lumberjack [50], a cooperation between the creators of rsyslog [53], syslog-ng [61] and Redhat was started on a Fedora conference with the goal to provide a complete CEE implementation. The effort resulted in rsyslog and syslog-ng implementing the complete CEE standard and Fedora supporting this effort as a first Linux distribution.

rsyslog

As being part of all these efforts to improve and enrich syslog's message format, rsyslog gained many unique features with regard to the ability to convert between all the different message formats. With the addition of liblognorm [31] rsyslog offers the possibility to extract structured data from log messages provided rules to convert the data are available.

By using an unstructured log format, applications aid in loosing information that was available at the point in time the log entry was generated but gets lost afterwards. Parsing log information to try to regain the lost information can only be regarded as a work around applications not doing proper logging.

3.4 Simple Network Management Protocol

At its core the Simple Network Management Protocol (SNMP) is organized around variables defined through Structure of Management Information (SMI) entities structured in a hierarchy in Management Information Bases (MIBs). Those variables may be accessed by a **manager** entity for reading or writing by contacting an **agent** that actually executes the read or write request. SNMP also has the ability to send notifications on certain events to a managing entity via a **trap**.

As a UDP based protocol, SNMP communication is stateless, reading and writing variables uses port 161 while trap comunication employs port 162. The main design criteria of the protocol were low overhead on the devices being managed or monitored and simplicity in use. The protocol itself is based on two distinct commands: one to read a variable and one to write to a variable. Later versions of the protocol also added bulk versions of those commands allowing to read from or write to several variables from a single command to make communication between manager and agent more efficient.

Protocol History

SNMP has its origins in the Simple Gateway Monitoring Protocol (SGMP) defined in RFC1028 [98] in the late 1980s that allowed polling of some important runtime parameters of a gateway. As stated in RFC1028, this protocol was meant to be an interim solution to gateway monitoring in a period of rapid growth of the then new internet [190].

In August 1988, RFC1065 [143], RFC1066 [142] and RFC1067 [97] were released that specify SMI, MIB and the SNMP version 1.

SMI, the definition of a datum, is defined using a subset of Abstract Syntax Notation One (ASN.1), a standard defined by the Open Systems Interconnection (OSI) gremium of the ISO as standard ISO8824 [121]. A SMI consists of a name, a syntax and an encodig where the name is referred to as Object Identifier (OID), the syntax specifies the data type used from both the basic data types of ASN.1.

OIDs cannot be freely chosen but follow a strict, tree-like, hierarchy. The topmost nodes include `org.iso.dod.inet` where "dod" is short for "Departement of Defense" clearly showing the roots in DARPA-Net of today's internet. Below the `inet` (internet) node of the tree all the data structures for SNMP start.

Note however that SNMP isn't the only protocol using ASN.1 for encoding and thus isn't the only protocol using this OID tree; LDAP, for example, is using that data format too.

MIBs use the numerical representation of OIDs to reference the position in the tree, which is for `org.iso.dod.inet.mgmt.mib`: `1.3.6.1.2.1`. The description of a node defines the type of access (read-only, read-write, write-only or not-accessible) and the details about its structure with respect to data type, its syntax and so on.

SMI and MIBs can be seen as semi structured data like XML documents described by a set of Document Type Definitions (DTDs) describing the structure and elements of the tree. The network protocol described in RFC1067 specifies how to access from the manager entity to an agent entity works.

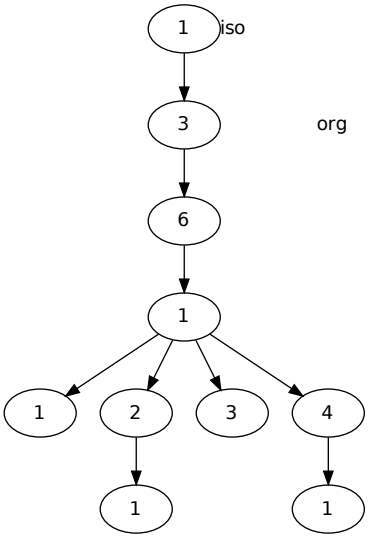

Figure 3.2: top of the OID tree

SNMP version 1 (SNMPv1) used a community string as sole authentication feature that was sent with each request. Soon after the release of RFC1067 work on improving the security of the protocol started but ultimately lead to 4 different versions of SNMPv2 mainly due to the inability of the involved parties to reach an agreement on the security model on which to base that new version:

- SNMPv2 or SNMPv2p ("p" for party) was defined in RFC1441-RCF1452 [80] [84] [85] [79] [107] [108] [141] [83] [86] [81] [82] [78] introducing a rather complex party-based security model; besides that allowing bulk requests was one of the major enhancements of the protocol.
- SNMPv2c as defined in RFC1901-RFC1908 [89] [92] [93] [88] [91] [94] [90] [87] is known to be the community based version of the protocol (thus the "c" appended to the protocol's name), that still uses community names for access control.
- SNMPv2u was defined in RFC1909 [140] and RFC1910 [188] and uses a user-based approach to authentication to also avoid the complexity of the original security model (the "u" attached to the protocol name indicates a user centric authentication model).
- SNMPv2* was a commercial derivative of the original SNMPv2 protocol.

The standardization of the original SNMPv2 in 1993 with its complexity lead to prolonged life and support of the older protocol, the development and standardization of SNMPv2c and the creation of RFC1908 [87] that describes the coexistence of SNMPv1 and SNMPv2c. Many

vendors use two community strings – “public” and “private” – to indicate access read-only and read-write access which are, in most deployments never changed. This ultimately lead to the perception of SNMP being an insecure protocol.

In 1997 an IETF working group was installed to continue the efforts on SNMP standardization and to create a single secure version of SNMP again. In 1999 the working group released RFC2570 [95] that provided a summary of the work done to reach a new protocol which was finally accomplished in 2002 with RFC3410 [96] and consists of the following specifications:

- RFC3411 [115], RFC3412 [77] and RFC3413 [137] describing the network protocol and its application.
- RFC3414 [74] describing a User-based Security Model (USM) for the protocol and introduces ciphers and hash specifications for the use with SNMPv3.
- RFC3415 [189] describing a View-based Access Control Model (VACM) allowing to specify access to certain attributes based on groups.
- RFC3416 [164] updating RFC1905 of SNMPv2 to clarify a few parts of the specification.
- RFC3417 [163] and RFC3418 [162] defining transport mappings and MIB specification updates in general.

From the very beginning it was obvious that the cipher and hash specifications will need to recieve updates from time to time: In 2004 the use of AES in SNMP was specified in RFC3826 [73], in 2009 the network transport subsystem of the protocol was redefined in a more modular way in RFC5590 [117] and RFC5591 [114] which lead to a new secure transport based on secure shell in RFC5592 [116]. Later efforts to improve transport security in SNMP included the proposal to use Transport Layer Security (TLS) to authenticate and protect SNMP traffic (RFC6353 [113] in 2011).

Current State

SNMP as a protocol is supported by almost any network gear available and thus a protocol every network monitoring and management stack needs to support to a certain degree. Although most devices implement SNMPv2c and SNMPv3, the crytographic options in most implementations include only AES with weak hash functions like MD5 and SHA1 as defined in the original RFCs. Newer ciphers or hashes either aren’t implemented or not specified: as of October 2015, a RFC defining SHA2 for SNMP is only on its way and it will take another couple of years until networked devices will have an implementation of those.

SNMP is easy to use and works very well for providing data about a network, ranging from package and error counts on switches to toner status in a printer to active information propagation by using SNMP traps. Still, the security of the protocol is lacking and will continue to lack in the future: protocol specifications and updates are far behind the current industry standard with regard to TLS/SSL and vendors are behind protocol specifications. SNMP, as most very old protocols, should probably only be used on small, isolated monitoring and management networks. Ignoring it, however, is not an option as the data that may be gained through it is too valuable to be lost.

3.5 statsd

Statsd describes a simple UDP-based protocol used for collecting and aggregating statistics. The original idea and implementation dates back to the work of Cal Henderson [18] of Flickr in 2008. His publication of that work [48] lead to a plethora of implementations [33] of that protocol [59] in many different languages.

The de facto standard implementation is the one by Etsy [22] that is feature rich in that it provides lots of plugins to connect to different backends and offers a tcp mode too. Nevertheless more specialized implementations for use with huge performance requirements like Github's Brubeck [8] exist. According to Brubeck's web page [7] it "has scaled to up to 4.3 million metrics per second (with no packet drops even at peak hours – as we've always intended)" where the Node.js implementation of Etsy according to the graphs on the same page seems to have suffered from packet drops of up to 22.000 packets per second at peak times.

The protocol itself is very similar to the standard syslog protocol [138] and therefore susceptible to the same kinds of problems: Confidentiality, Integrity and Authenticity are missing (see syslog evaluation in 3.3). Brubeck's authors seem to have partly addressed these issues by implementing statsd-secure [8]: "like StatsD, but each packet has a HMAC that verifies its integrity. This is hella useful if you're running infrastructure in The Cloud™©and you want to end back packets back to your VPN without them being tampered by third parties". By the suggested use of VPN and the use of a shared secret for integrity verification, only the completeness of the data is in question. This even more as UDP just drops packets it cannot handle which may lead to completely wrong data.

3.6 Network Monitoring Software

Protocols like the aforementioned SNMP do require to have a manager software at hand that allows to make use of the protocol to acquire the desired data. The very same holds true for the syslog protocol and the tools introduced up to now in this chapter. Frameworks to collect and present data using these tools and protocols will be introduced in the course of this section.

Multi Router Traffic Grapher

The Multi Router Traffic Grapher (MRTG) [153] collects network statistics from network devices like routers or switches using SNMP, feeds that data into Round Robin Databases and generates graphs from that data. Besides pure data collection, MRTG is also able to send out alerts in case a configured threshold condition was met. An alert is sent via email to a preconfigured email address. Note that the I/O load is rising proportional with the amount of network interfaces or numbers being collected and monitored. The storage backend needs to be planned accordingly.

Collectd

Although there are tools for servers making performance data available via SNMP, the use of that protocol isn't necessary as many other, better fitting tools for data collection on servers are

available. One of these tools is collectd [17] that provides a huge set of plugins to gather all kind of different metrics from hardware and software running on a server. One of collectd's design criteria was to keep resource usage and dependencies minimal; to accomplish that, collectd uses RRD databases for storing metrics only and leave the graphing and analysis to other tools. Only a simple alerting feature similar to that of MRTG is available.

Implemented in C with minimal dependencies greatly reduces the overhead of gathering data. For this reason, collectd also allows detailed monitoring of embedded devices and is included in distributions like OpenWRT [47] that are used in, for example, wifi access points.

Besides providing local data collection, collectd also provides a mechanism for data collection via network. Since May 2009 with the release of version 4.7.0, collectd's network protocol gained support for signing and encrypting networked traffic including the ability to enforce signature or encryption for communication. The foundation of these features lays on an authentication mechanism that holds passwords on the server which are used as secret key. The builtin network stack thus requires out of band account management including distribution of passwords to the different hosts.

Ganglia

Ganglia [24] is a monitoring tool designed for clusters and grid computing. Organized in a hierarchy it offers federation over wide area links and uses either a udp-based multicast protocol or a tcp-based protocol for data exchange. The only security mechanism applicable to the network protocols are access control lists to the tcp-based protocol.

The idea behind that seems to be the origin in grid computing where loaded systems aren't considered to be a secret but a regular use case. Most of the time, compute clusters aren't directly attached to a public network but share a private cluster network infrastructure used for monitoring communication too. Thus Ganglia isn't a general purpose monitoring tool.

Nagios

With its origins dating back to 1996 and the first release as NetSaint [39] in 1999, Nagios [40] is one of the oldest open source monitoring tools that is still under active development. Nagios, as all its derivatives, is a central monitoring system designed to schedule checks of all monitored systems and services at regular intervals ("active checks"). It is, however, also possible to configure Nagios in a way that external applications are allowed to submit check results to a Nagios instance ("passive checks"). Based on check results Nagios deduces a host and service state and alerts administrators accordingly. Over time Nagios gained features beyond monitoring that include statistics collection, failure prediction, syslog message integration, parallel host and service checks and service dependencies. These features are added as plugins to the core Nagios system; some of which are only available as commercial plugins.

Nagios itself contains a job scheduler and a result collector while service and host checks are done by plugins which also are a valuable resource for other monitoring systems [37].

History

Between 1999 and 2006 Nagios was developed by a community as an open source project lead by Ethan Galstad, the original creator. While Nagios gained more and more attention and was even selected by SourceForge.net as the project of the month in June 2005, the need for professional consulting and support services arose. Ethan Galstad founded "Nagios Enterprises, LLC" in 2007 to provide commercial services. Ultimately this resulted in the release of "Nagios XI", a commercial distribution of Nagios, and the renaming of the original open source version of Nagios into "Nagios Core" in 2009.

The community perceived the foundation of a company as a massive slow down on the development of the open source version. The major annoyment in Nagios deployments at that time were the limitations of the check scheduler engine that did not scale well. As a result, Nagios was forked [106] into a new project named Icinga [29] in 2009. Besides technical aspects, trademark issues and personal discrepancies played a major role in the process of that fork [147]. For Nagios, the development situation improved as two new core developers, Andreas Ericsson and Ton Voon, were added to accompany Ethan Galstad in his role as the project's gatekeeper.

Years later, in 2013, Andreas Ericson announced at the Open Source Monitoring Conference (OSMC2013) [46] that he was removed from the core developer team after writing about 95% of the code of the latest major release of Nagios and that thus he is forking Nagios in a project named Naemon [38].

Technical Details

When talking about Nagios one usually refers to "Nagios Core" the daemon that offers the following features:

- check scheduler
- event broker
- notification service
- plugin interface

Through the plugin interface, service checks may be integrated using Nagios Remote Plugin Executor (NRPE) [43] or Nagios Service Check Acceptor (NSCA) [44] or other means of transport like a Secure Shell (SSH) transport for plugin execution [42].

Other annoyances of Nagios deployments besides performance issues of the scheduler include the use of a monolithic configuration file that becomes the harder to manage and update the more hosts and service checks are added and the use of a single `status.dat` file that gets updated by all check results and is meant to be used by several external components as an API to get access to state information.

"Nagios Data Out" (NDO) [41] is a plugin that works around some of the problems arising from the use of a single file as an API for accessing status information. The use of NDO itself, however, introduces new issues like that accessing data from Nagios may block the core or that

NDO has to run housekeeping jobs to clean up old data which are quite expensive in terms of CPU usage.

Security handling within the Nagios community does not work too well: In September 2009, a bug [120] against Nagios' NRPE plugin has been filed showing fundamental flaws in the protocols security:

- no authentication
- no client or server verification
- keys are being generated at compile time, which means that all installations using packages use the same key material.
- Anonymous Diffie Helilman cipher is hardcoded in the library

The bug report also contained a patch claiming to fix these issues and finally on October 27th 2015 someone from the core team decided that this bug is worth a fix by announcing to start working on a fix.

Together with other bugs allowing arbitrary command execution [112], the main mechanism for doing remote checks in Nagios, NRPE, is unusable except for some isolated monitoring networks – and more: it puts all machines on a network in danger and should be avoided. This basically leaves plain Nagios only with the SSH-based approach for executing remote status checks, which has a problem on its own: SSH allows full shell access to a system, thus whoever is in control of the monitoring server, also is in control of all devices monitored that way. And worse: given the speed in security incident handling, it is very much likely that there are other security relevant bugs hidden somewhere in Nagios allowing to take over the central monitoring server.

All of Nagios' security issues [19] do potentially also affect all forks as all vulnerabilities in plugins (like the vulnerability in NRPE) do possibly affect all applications using these plugins.

Shinken

In an effort to improve Nagios' performance and experiment with a new, multithreaded architecture, Jean Gabès implemented a prototype of a possible new software design for Nagios in Python he named Shinken [105]. After some discussion about the concept with Nagios core developers, he decided to continue his project as a reimplementation of Nagios in Python [56].

While Shinken is no fork like the aforementioned Icinga and Naemon it still is a Nagios derivative that remained compatible to the original Nagios toolchain and is even able to use a Nagios configuration file.

Architecturewise Shinken splits up the work into several different processes of which most may be parallelized to scale for bigger deployments. The central component is the Arbiter responsible for parsing the configuration, starting the other processes and supervising them. Scheduler processes take care of the execution of the checks while Poller processes actually run these checks by using transports like NRPE, SNMP or CommandPipe, a mechanism that allows receiving results of passive host checks. Schedulers do have a simple and robust design

and thus do only get the parts of the configuration that is relevant for the workload they're chosen to handle by the Arbiter: the checks they're responsible for. Event and notification handling is done by the Reactionner. The Broker is the component that allows interaction with external components like the native web interface, graphite, several databases or other additions to Nagios' core.

Check_mk

A completely different approach to work around performance issues in Nagios check scheduler is that of Check_mk [12] by Mathias Kettner. The basic idea is to use a single agent on every monitored server that is capable of running all service checks required for that host. This lowers the number of checks to be scheduled by Nagios to one per host. The check_mk check then submits every single result as a passive check to Nagios.

Check_mk also offers some very convenient features like the automated creation of a Nagios configuration based on the check results of a host (Check_mk CCE [10]), a module to access the current status of servers and services without the use of `status.dat` or the aforementioned NDO via a UNIX socket called "Livestatus" [13].

Furthermore a new web interface, Multisite [15], that allows the definition of custom views and the integration of of several additional tools. Business Intelligence (BI) [9] is a tool that allows modelling dependencies between servers and services for availability calculations and alerting and the Event Console [11] is a tool that consists of a daemon that is able to collect syslog data and SNMP traps and filters, aggregates classifies them according to user specifyable rules.

The commercial edition of Check_mk even provides its own scheduler core [14] and thus is able to completely replace Nagios while maintaining its full functionality.

Check_mk's network protocol [16] can be considered rather secure in the sense that no commands can be injected: the agent simply does not accept any arguments and the server's parse work is rather light. The connection itself still needs to be protected by either a SSL tunnel and limited to IP addresses allowed to trigger status checks as the protocol itself embraces no means of authentication.

3.7 Smaller Tools

Linux and UNIX systems offer a plethora of simple single purpose monitoring tools that supervise RAID arrays, check for security updates, watch disk usage, oversee hardware or surveil services. Reporting in most cases is done by sending an email to the administrator in case of an event requiring intervention.

Sysstat

A versatile system statistics collection tool specifically designed for Linux systems is Sysstat [110]. It offers access to detailed information about CPU and memory usage, processes and their resource allocation and statistics about the I/O subsystems like disks, tapes or networked file systems.

Besides interactive commandline tools showing current values, Sysstat is also able to collect these values for later analysis into a file. A periodic job running by default every 10 minutes collects and stores all the metrics. The data almost directly come from the Linux kernel that offers a rich amount of counters and performance data. Sysstat picks up the data through the kernel, normalizes it and calculates the values of interest.

Many tools that are able to parse and graph the data collected by Sysstat or convert it to RRD or Whisper to use their toolstack for graphing. Moreover Sysstat is able to convert its databases into some common data formats like CSV, XML or JSON.

Sysstat does not provide any alerting features, its scope is to collect statistical data about a system for archiving or interactive analysis.

Smokeping

Smokeping [155] is a tool that monitors host availability and network latency by sending out requests to the host and measures the time until a response is received. The default protocol used for probing is ICMP ping but many other protocols may be used for the same purpose, including SSH, HTTP, FTP or DNS.

By default, Smokeping sends 5 ping requests every 300 seconds, collects the responses and their latency and stores them in RRD files. All hosts and the corresponding tests may be individually configured. The data can then be graphed with the help of rrdtools and interactively explored with the help of a CGI script that is provided with Smokeping.

In case of certain latency patterns or the inability to reach a host, Smokeping may send out alert messages via email to pre-defined email addresses.

Measuring latency on a link can only be done between two devices that are connected through that link. Smokeping offers a master/slave mode [156] that allows to run measurements from other hosts and collect their results: The master that also runs the web interface provides configurations containing checks and targets to slaves. Slaves authenticate themselves to the master by using a predistributed passphrase with HMAC-MD5 as defined in RFC2104 [136], fetch the configuration, run the checks and report back the results. The communication may be additionally secured by using SSL.

There, however, is a significant drawback in this network mode that makes it impossible to share smokeping instances with others: the configuration is exchanged in the form of perl code that will be executed on the slave. Thus, everyone allowed to send configurations to a host is able to execute arbitrary commands on the slave.

3.8 Summary

The evaluated protocols and tools provide capabilities to collect, aggregate, store or evaluate log information and status data of services, servers and other devices in a network. Table 3.8 shows the tools' abilities with respect to the criteria mentioned in the introduction to this section. Based on these criteria some interesting observations for developing a new approach to system monitoring can be made.

	collect	transport	process	store	present
RRD			X	X	X
whisper		(X)	X	X	X
syslog	X	X	(X)	X	
SNMP	X	X			
statsd		X			
nagios	(X)	(X)	(X)	X	X
check_mk	(X)	X	(X)	X	X
sysstat	X		(X)	X	(X)
smokeping	X	(X)	(X)	X	X

Table 3.5: evaluation of monitoring systems

Network Communication and Security

Collectd and Smokeping are the only tools providing sufficiently secure network communication out of the box. The simplicity and low overhead of Collectd's cipher is acceptable considering the light resource usage. Smokeping takes a different approach that inverts the direction of communication: the slaves connect to the master to fetch their configuration and to submit probe results. The effect on security is that no open network ports on the slaves need to be protected, only the master server needs to provide its service via SSL and the slaves are required by the protocol to authenticate to the master. It is, however, unfortunate that sharing Smokeping Slaves between different parties isn't possible in a secure way.

For the Syslog protocol, secure means of transport are available but require extra configuration and and support from all devices that need to exchange data.

Tools like Nagios, Icinga, Shinken, Naemon or Check_mk require the use and correct configuration of external tools to allow secure communication over a network. Check_mk is the only tool providing a page mentioning how to accomplish a secure deployment of the tool. All the others even advertise their support of a long known to be broken protocol – NRPE – as a feature although they have bug entries in their bug trackers about the security problems with NRPE.

Architecture

All tools evaluated do not only favor a centralized monitoring approach but also encourage to centralize test execution. Nagios with its weaknesses in the check scheduler suffers from scalability issues due to its excessive central check triggering. Check_mk showed that the load of the scheduler can be reduced tremendously by just triggering a master check per server that does all single service and status checks and reports them back at once.

A less centrialized approach may even more reduce the load of a central monitoring device: Triggering the check runs locally and just submitting the results, similar to Smokepings Master/Slave Mode, to the main monitoring server would even further reduce the load and immediately remove the need for the insecure NRPE. As a consequence the monitoring server then

needs to keep track of outstanding check results.

"Push operates on a key assumption – that it is possible to forecast or anticipate demand."

John Seely Brown, The Power of Pull: How Small Moves, Smartly Made, Can Set Big Things in Motion

CHAPTER 4

XMPP

The eXtensible Messaging and Presence Protocol (XMPP) is a secure, decentralized and extensible realtime communication protocol based on XML streams over an encrypted and authenticated network connection. Originally the protocol was called "Jabber" by its inventor Jeremie Miller during its initial development in 1998 and 1999. While being standardized through the Internet Engineering Task Force (IETF) in 2002, it was renamed to its current name: XMPP. In october 2004, the standardization process resulted in RFC3920 [167] and RFC3921 [168] which together form the foundation of the protocol known as XMPP Core and XMPP Messaging. Around 2009 the process to update these RFCs, clarify parts of the protocol and add new error codes started which in march 2011 resulted in the release of RFC6120 [170] and RFC6121 [171] that obsoleted the older RFCs. Finally in June 2015 RFC7590 [174] was released that added more security constraints with regard to the security standards of XMPP connections.

Beside the core of XMPP that is defined in RFCs, the XMPP Standards Foundation (XSF) – the governing body of the protocol – has its own process of defining extensions to the protocol written and improved in quite a similar way the IETF defines RFCs that results in XMPP Extension Protocol (XEP) [166] definitions. Some very important parts of the protocol are based on such extensions such as server side components as used for multi user chat (MUC) or ad-hoc commands, both of which will be examined in the course of this chapter.

4.1 Other Messaging Protocols

XMPP borrowed some ideas from protocols predating it; understanding some of the inner workings of XMPP is easier when understanding problems and solutions in other protocols with a similar scope. Interestingly, some of these messaging protocols have a history of being used for management or monitoring of computer systems.

Many proprietary solutions like AIM [1] (AOL Instant Messaging), ICQ [30] or Skype [57] exist that are based on a centralized infrastructure. Although many those protocols support encryption at the network layer, those protocols offer chat or video chat services only and do not prove to be extensible in the way XMPP is.

Simple Mail Transfer Protocol

Besides the Domain Name System (DNS) [177], the Simple Mail Transfer Protocol (SMTP) is one of the oldest standardized protocols in the internet that is still in use today. The original standard was released as RFC788 [160] in 1981. RFC821 [161] followed in 1982, RFC2821 [134] updated the specifications in 2001 and to date RFC5321 [135] released in 2008 describes the current standard.

SMTP is, like syslog described in 3.3, a very old protocol that does not have many security features built-in. Messages being sent with this protocol may be relayed through different servers of which any adds its own envelope header in a way that the path a message takes can be traced. As the messages travel their path in plain text with no cryptographic hash whatsoever, they may be modified tracelessly anywhere along the path; even the path information itself can be freely changed. Email addresses provide unique identifiers that give routing hints to the servers. Such an address – for example `user_a@example.com` – consists of a local part – `user_a` – and a domain part – `example.com`. The domain part is the global part in the addressing scheme as domain names are backed by the Domain Name System (DNS) and point to the corresponding servers that are responsible for delivering a message to its recipient.

Although SMTP provides a built-in reliable message transport – a user always gets feedback from a server in case a message could not be delivered – message delivery does not have to be instant and instant delivery cannot be forced due to an unpredictable amount of servers that may be used for relaying a message. The protocol itself also only handles the delivery of a message to a destination server, for a user to pick up the message, different protocols like POP3 or IMAP exist. SMTP is a "federated protocol" which means, that anyone owning a domain name may run an own SMTP server that serves the owner's domain and may exchange messages with everyone else who also operates an SMTP server. Even more so: every SMTP server may relay messages for any sender; there is no limitation in terms of specifications. Only today servers choose which messages they are responsible for.

In the beginning of SMTP the focus for designing the protocol was on reliability and flexibility: many users – be it companies, people or organizations – only had dial up connections and were not permanently online; being able to relay messages via other servers on the network that were able to queue messages until a destination server became online was a very useful feature. Also being able to send mails with ones own email address as the one of the sender via the mail server of the dial-up line internet service provider was a requirement for the success of the protocol.

Today SMTP suffers from a huge spam problem mainly due to the absence of basic security features within the protocol itself. Unfortunately, adding these features to the protocol will break it and there were already many approaches to curtail some of the problems while leaving the core features of the protocol intact which all ultimately failed to fix the protocol; efforts to replacing SMTP with a different protocol to date failed too.

SMTP in system monitoring

The use of SMTP in system management and monitoring is ubiquitous as the protocol is a common denominator for pushing out notifications about events on systems, even more on legacy

systems, because SMTP is available as a transport for notifications on most devices, ranging from printers to storage systems and, of course, for all kinds of desktop and server systems.

For Linux server systems, one of the most common mail server implementations used to forward mail notifications to the system's administrator – Nullmailer [45] – finally gained support for using transport security in 2012 with the release of version 1.10. The main issue here is that supporting SMTP as a protocol does not allow any conclusions about the support of the more advanced and desperately needed features of SMTP like transport security or authentication, as those parts aren't mandatory.

Internet Relay Chat

Internet Relay Chat (IRC) predated XMPP by about 10 years: Jarkko Oikarinen of the University of Oulu in Finland implement the first client and server in 1988. Originally meant as an extension for real-time commenting in Bulletin Board Systems (BBSs), IRC quickly turned into the defacto standard for live chat systems [28]. In 1993, IRC was standardized in RFC1459 [157] but many of the server implementations did not quite follow the standard. RFC2810 [128], RFC2811 [129], RFC2812 [130] and RFC2813 [131] tried to bring on track all the different implementations in 2000. RFC7194 [118] from April 2014 that contains some notes about SSL/TLS and port usage sums up that effort like this: "For details on how IRC works, see [RFC1459], [RFC2810], [RFC2811], [RFC2812], and [RFC2813]. Please note that IRC is extremely fragmented, and implementation details can vary wildly. Most implementations regard the latter RFCs as suggestions, not as binding."

While being the first widely available instant messaging system, it suffers from several weaknesses compared to XMPP:

- IRC does not support open or standardized federation which means that a user needs to choose which IRC network to join to be able to meet the people he wants.
- The authentication model is weak to non-existent: A user may choose any nick name he wants, provided the name isn't already taken.
- On the network layer all communication is, by default, unencrypted although several variations of encryption support do exist, end-to-end encryption cannot be enforced.
- Due to the nature of the protocol layed out in RFC1459 (and the later RFCs), basically only text exchange is possible.

IRC in system management and monitoring

As IRC was very popular among system's administrators, the usage of IRC bots – scripts or simple daemons listening on an IRC channel for instructions or sending out system health messages – was very wide-spread. Even some IRC bots existed that sent commands to servers. Later on, many Command and Control (C&C) servers for malware used IRC as their main communication protocol.

XMPP Based Messaging Services

Many popular push notification services today are based on XMPP technology, like for example "Apple Push Notification Service" [3] (APNs) or the notification service "Google Cloud Messaging" [25] (GCM) for Android. Most of these implementations either added their own proprietary extensions or do not allow federation.

Google Talk

Several publicly available services use or used XMPP as their core protocol with the first being Google Talk [26], a chat and telephony service introduced by Google in August 2005. The service federated with all XMPP servers and made Google Talk an open service. After the announcement of its successor Google Hangouts in 2013, federation was finally disabled in 2014.

Google Wave

Google announced a new kind of service designed to finally replace SMTP on its yearly developer conference in 2009: Google Wave. The foundation of that service was the XMPP protocol, even federation was meant to be built-in. The main concept was to organize conversations in Waves that could be enriched with more specialized services like polls or with multimedia content. Several companies like SAP announced support for Google Wave [54] in their products but in 2010 Google announced, to the surprise of many, the disconinuation of the service in 2012. The source code was opened up by Google and lives on as Apache Wave [2].

Facebook Chat

In 2010 Facebook opened up its chat service [23] via XMPP mainly for use in applications. Although using XMPP for communication, the service itself was never opened up by allowing federation. Beginning with 2014, the service was abandoned.

WhatsApp

WhatsApp is a popular instant messaging system for mobile devices introduced in 2009. At its core WhatsApp uses a proprietary modified variant of XMPP [65] based on a very popular open source XMPP server. In 2012 many security issues with WhatsApp were discovered [64] [27] that were subsequently fixed; the most pressing issue was unencrypted network communication that allowed reading along other's messages and even taking over foreign accounts.

4.2 Naming Conventions in XMPP

Similar to SMTP, the XMPP protocol uses Jabber IDs (JIDs) which are, just like email addresses, unique through its domain name and even look similar to them: `userA@example.com`[1].

[1] People in the XMPP community seem to have a faible for Lewis Caroll's "Alice in Wonderland" and continue to use names and places of the story for their protocol definitions. This thesis will not follow that convention and will

Jabber IDs

A JID like `userA@example.com` is called a **bare JID**. It consists of a local part and a domain part and is used for general addressing. When a user logs in to a XMPP server, a resource is added to the bare JID allowing the server to distinguish several sessions of a user and route messages and responses to the correct session. Such a **full JID** may look like this: `userA@example.com/desktop` where `desktop` is the resource identifier. With the help of a "priority" a user using several sessions may notify the server about the primary client new messages should be routed to.

The uniqueness of a bare JID is the foundation for **federation**: every domain may operate its own XMPP server for its users allowing them to communicate with users from other domains and vice versa. Unlike with IRC, even features like multi user chatrooms (MUC) or news services on other servers may be used that way.

Connections

Unlike SMTP, XMPP is not transient: A client talks to his server directly and the server talks directly to the other user's server; relaying of messages through multiple hops inbetween is neither possible nor desired in XMPP – this, in essence, avoids spoofing of messages because a server will only accept messages from a server authoritative for a domain. In the SMTP world, message spoofing is part of the protocol specification and one of the building blocks of spam. XMPP servers implement two similar but different protocols: Client-to-Server (**C2S**) and Server-to-Server (**S2S**). The main difference between those is authentication: clients proof their identity to servers by providing authentication credentials (username and password in the most common case), while inter-server authentication has several different mechanisms with server dial-back being the most common one. Transportation of messages works the same on both types of connections.

S2S authentication

Regarding basic security aspects discussed in 2.1, authentication between servers is a crucial factor when using inter-domain communication without any additional layers of security like message encryption: every message in that scenary passes two servers only: the own server and that of the other party, provided the authentication between these two worked.

Server authentication in S2S is possible in mainly two ways: the preferred one is by using certificates signed by a trusted third party (a certificate authority) and the older method is called Server Dialback as specified in XEP-0220 [125]. Unfortunately even using certificates signed by a generally trusted certificate authority does not provide ultimate security as the trust model behind certificate authorities is largely broken [165]. Additional techniques like pinning certificates (remembering fingerprints of certificates and comparing them with those of the current session), adding certificate fingerprints to DNS and, finally, using DNSSEC [69] [71] [70] to add more trust to the results of DNS queries.

---

use the standard `userA@example.com`.

On the other hand, compared to other federated protocols, XMPP is the only such protocol that from the very beginning had authentication of server to server connections built into the protocol. RFC6120 [170] states on page 169: "Before RFC 3920 defined TLS plus SASL EXTERNAL with certificates for encryption and authentication of server-to-server streams, the only method for weak identity verification of a peer server was Server Dialback as defined in [XEP-0220]. Even when [DNSSEC] is used, Server Dialback provides only weak identity verification and provides no confidentiality or integrity. At the time of writing, Server Dialback is still the most widely used technique for some level of assurance over server-to-server streams."

Presence and Rosters

A user's contact list stored on the server is called a **Roster** which is also used to keep track of the user's presence subscriptions. **Presence**, a core feature of XMPP as a real-time commuication protocol, is relevant to the user's pricvacy: only users allowed to see a user's presence status also gets presence information about a user forwarded by the server. The process of establishing such a trust relationship is called **Presence Subscription** in which a user sends a presence message of type `subscribe` to another user who may positively answer that request with `subscribed` and vice versa. The presence subscription information also needs to be updated and reflected in the user's roster.

Stanzas

XMPP connections are long living connections transporting streams of XML data. These short streams or communication fragments are the building blocks of the protocol called **stanzas**. Three main types of stanzas do exist:

`<message/>`

`message` stanzas provide a simple way to push information from the sender to the receiver. This kind of stanza usually does not provide means of acknowleding the delivery. There are several types of messages qualified by the `type` attribute:

- `normal` – indicating just a single message sent one way, somehow similar to an email message. A reply may be sent, however.
- `chat` – a session-like real-time message exchange between two entities.
- `headline` – just a one way message with no reply at all.
- `error` – any entity detecting an error with a previously sent message will return a message with type error.

The following shows a test message of type chat as received by `userb@example.com`:

```
<message from="userA@example.com/desktop"
         to="userb@example.com"
```

```
        type="chat">
    <subject>Test Message</subject>
    <body>This is a test message...</body>
</message>
```

The `from` attribute will always be filled out by the server to avoid message spoofing. Note that messages normally do not get lost in XMPP communication; at least not as easy as UDP packets on a network like with the syslog protocol or with statsd.

<presence/>

Exchanging presence information is vital for real-time communication and allows to see what other entities are online at the moment. Presence information is sent to the server with no recipient (and no sender to avoid spoofing), because the server stores all user's relations to each other and forwards presence notifications to everyone allowed to see them. This is an example stanza as recieved by `userb@example.com` who has permission to see user A's status:

```
<presence from=userA@example.com/desktop}
    <show>xa</show>
    <status>ready to talk...</status>
</presence>
```

As already mentioned in 4.2, presence stanzas may also contain type information (`subscribe`, `subscribed`, `unsubscribe`, `unsubscribed`) that adds or removes other users from receiving presence broadcasts. Unsolicited presence stanzas of type `subscribed` or `unsubscribed`, i.e. confirming an unrequested action, are discarded by the server because they were not requested by the client.

<iq/>

IQ is short for Information/Query, a stanza type for more structured request/response interaction. The type attribute for this kind of stanza has to be one of those:

- `get` – request information from the server or another entity on the network.
- `set` – change or update information on the server or provide information for another entity on the network.
- `result` – positive reply acknowleding a successful `set` operation or containing the results of a `get` operation.
- `error` – negative reply of an entity stating that it was unable to fulfill a request.

This kind of stanzas may be used for roster updates on the server:

```
<iq from="userA@example.com/desktop"
    to="userA@example.com"
    type="set">
    <query xmlns="jabber:iq:roster>
        <item jid="userb@example.com"/>
    </query>
</iq>
```

Note that it is always the server that is responsibe for processing requests, no matter of what type, if the bare JID is in the `to` attribute: in the above example, the server needs to add a contact to the roster, in case of a simple message, the server has to decide how to deliver a message.

4.3 Protocol Extensions

Building upon the foundations of the protocol, the extensions defined in XEPs are what distinguishes XMPP: currently there are more than 170 such features in a published state, some of which being a building block for many others, some rather smaller enhancements and others may end up being rejected by the XSF. Many of those extensions specify either a client- or server-only feature and not all clients or servers do support all extensions.

During the process of standardization [166] such an extension has to go through different states as illustrated in Figure 4.1. Besides being formally and technically correct, an extension also needs to be used within its scope: some extensions were rejected because there were not the required two independend implementations available. It is of course possible to develop and use private and unpublished extensions but that also means that such an extension will not get extensive review and feedback by the community and that one has to do all the implementation work on his own.

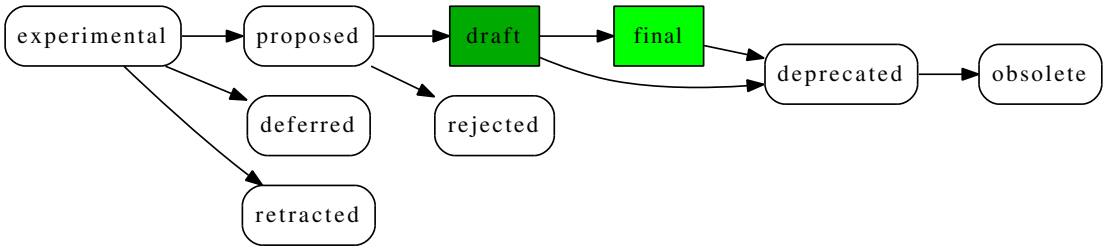

Figure 4.1: states of standardization

Service Discovery

Entities on a XMPP network may be diverse: humans using a chat client, bots, servers, chat rooms and so on. In the same way the number of protocol features they support is different: The service discovery protocol as specified in XEP-0030 [119] (and later extended and clarified

in XEP-0128 [169]) helps with discovering entities in the XMPP network and with finding out what they do.

```
<iq ...
    to="example.com"
    type="get">
    <query xmlns="http://jabber.org/protocol/disco#items"/>
</iq>
```

There are basically two types of queries, both within `<iq/>` messages:

- `items` – other entities available through the queried entity, just as the answer to the above request:

  ```
  <iq from="example.com"
      ...
      type="result">
      <query xmlns="http://http://jabber.org/protocols/disco#items">
        <item jid="conference.example.com"/>
        <item jid="pubsub.example.com"/>
        (...)
      </query>
  </iq>
  ```

- `info` – after the discovery of other entities, more details about an entity may be found by issuing a query of this type by using the namespace `http://jabber.org/protocols/disco#info`. A reply may look like this:

  ```
  <iq from="conference.example.com"
      ...
      type="result">
      <query xmlns="http://http://jabber.org/protocols/disco#info">
        <identity category="conference" type="text" name="Chatrooms"/>
        <feature var="http://jabber.org/protocol/disco#info"/>
        <feature var="http://jabber.org/protocol/disco#items"/>
        <feature var="http://jabber.org/protocol/muc"/>
        (...)
      </query>
  </iq>
  ```

Service discovery is essential when navigating in a XMPP network. Users, of course, may not be discovered in this way: a trust relation ship between the entities querying each other has to be established first (see the section on Rosters and presences subscription4.2 for details).

Event Logging over XMPP

XEP-0337 [186] describes a special message format to be used for event log messages and the accompanying discovery of that feature. The extension itself is rather young and still is in experimental state but fits well for a accomplishing some tasks in this thesis. In service discovery this feature appears as `<feature var='urn:xmpp:eventlog'/>`. It describes an extension to the regular `<message/>` type by defining a new namespace `urn:xmpp:eventlog`. The root tag of such a message is `<log/>` which requires a mandatory `timestamp` and an embedded `<message/>` but many other fields and tags are specified in the XEP that allow many different use cases ranging from embedding plain syslog messages to detailed logging of stack traces:

- `<log/>` allows the following attributes:
  - `timestamp` – a required attribute that contains a time stamp in the same format as it is used in RFC5424 [109] for syslog.
  - `id` – a mean to classify an event. It is recommended to use a single string to help with automatic grouping and filtering of events. Examples given in the XEP include "Login" or "ConnectionProblem".
  - `type` – corresponds to syslog's severity levels: `Debug`, `Informational`, `Notice`, `Warning`, `Error`, `Critical`, `Alert` and `Emergency`.
  - `level` – in addition to syslog's severity levels `Minor`, `Medium` and `Major` may be used for further classification.
  - `object` – for specifying the object of an event.
  - `subject` – to describe who triggered an event.
  - `facility` – may be used to map syslog's facility parameter or, in a more literal sense, describe the facility in the network or on the system.
  - `module` – the module in a larger system stack where an event happened.
- `<stackTrace/>` – a text field to hold a stack trace.
- `<tag>` – an element that may appear as often as needed that holds key-value pairs:

```
<tag name="sender" value="user1@example.com"/>
<tag name="recipient" value="user3@example.com"/>
<tag name="msgid" value="1ZRuvv-0008U2-0N"/>
```

- `<message/>` – a mandatory text field for the event message.

As already mentioned above, this `<log/>` structure only requires `timestamp` and `<message/>` to be used; all the rest of the elements and attributes are optional but very helpful to keep already parsed information. An example of an already parsed syslog message fitted into the `<log/>` data type:

```
<message from="servera" to="log@syslog.servers.example.com" type="normal">
      <log xmlns="urn:xmpp:eventlog" timestamp="2015-08-10T12:22:31Z"
           id="BruteForcePassword" type="Warning" level="Minor" object="ubnt"
           subject="10.10.1.10" module="auth">
         <message>Failed password for invalid user ubnt from 10.10.1.10
                  port 52395 ssh2</message>
      </log>
   </message>
```

As log information potentially contains sensitive information, the destination for such messages should be thorrowly selected but this is beyond the scope of a XEP.

Publish-Subscribe

Up to now, all messaging mechanisms examined work with direct addressing – a JID sends a message to another JID – with one exception: presence notifications. And more, users establish a relationship between them by *subscribing* to each other's presence. Hence the server automatically pushes updated presence information to all subscribers.

Based upon this simple mechanism a powerful and more generic implementation of the concept of Publish-Subscribe (PubSub) is described in XEP-0060 [145]. This fundamental mechanism is considered to be a building block of XMPP.

PubSub is organized around *nodes* to which information gets published. JIDs may subscribe to these nodes to get immediately notified in case of new information arriving. Addressing the PubSub extension of a server can usually be done by adding "pubsub" or "notify" to the domain part of the JID: `pubsub.example.com`.

Note that most regular chat clients are unable to handle or display PubSub messages without additional plugins.

PubSub Node Management

Creation and deletion of nodes is done in a simple `<iq/>` stanza using either `<create />` or `<delete />` as command payload and the node name as an argument. Based on server permissions, a user may or may not be allowed to create such nodes; assuming correct server permissions, node creation can be done with such a message:

```
<iq from="servera" to="pubsub.example.com" type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <create node="test" />
  </pubsub>
</iq>
```

In case of a `<delete />` all users subscribed to a node will receive a notification.

After creation, there are several important configuration decisions to be made with regard to who may access a node, who may publish to a node or how published items get delivered.

PubSub nodes offer many different means of access control for regulating access to a node; still, users have to subscribe to the nodes the want to see. The `access_model` only defines who is allowed to subscribe:

- `open` – anyone may freely subscribe to a node; the node is completely public.
- `whitelist` – a given list of JIDs is allowed to subscribe to a node.
- `authorize` – anyone may request subscription but the node owner has to approve or reject the subscription.
- `presence` – limits subscriptions to those who already subscribed to the node owner's presence in any way (`from`, `to` or `both`) is allowed to subscribe to the node.
- `roster` – limits those allowed to subscribe to the members of a group of the node owner's roster.

Closely related to the `access_model` is the `publish_model` which even interferes a little with the first and describes who is allowed to publish to a node. Available options are

- `publishers` – a list of JIDs allowed to publish is specified with the node owner by default being one of them.
- `subscribers` – every node subscriber is also allowed to publish to the node.
- `open` – anyone is free to publish to a node.

Everyone on the list of publishers is allowed to subscribe to a node, no matter what the `access_model` is. A node owner has full control over a node, whereas a publisher may publish to the node and subscribe to it and a subscriber only has viewer rights.

PubSub Message Handling

Several options regarding the items to be published to a node need to be considered too. The most important one being whether the node should store a history of items – `persist_items`, which is a boolean value – and if so, how many items (`max_items`) the node should keep. During the specification of PubSub this topic lead to heated discussions between the party advocation a pure message passing system vs. those advocating flexibility in what should be possible with such a system. It was finally decided to support both options and let the users choose which model they prefer. Nodes that store items work like a FIFO where the oldest item in a node automatically gets overwriten by the item just arriving when the number of items matches `max_items`.

The node's ability to store items also lead to another interesting option, namely to `deliver_payloads`, an option that allows to store the item in the node and only send out a notification to subscribers about the availability of the item with its ID. The subscribers then may choose to fetch the complete item when they are ready to. This may be especially useful when the expected size of an item – configureable with `max_payload_size` – may be huge.

For a monitoring system, such a mechanism may be used to collect check results: receiving periodic notifications about the availability of check results of a system (assuming one PubSub node per system) hints at basic availability of the system; further details are available when the monitoring system fetches and analyzes the item containing detailed check results.

Adhoc Commands

XEP-0050 [146] specifies a mechanism for command execution on a remote client, named adhoc commands. There is no requirement for any client to implement any command and such an adhoc command isn't tied to a JID but to the client software implementing such a feature, thus having adhoc in the name. To detect the possibility of running such adhoc commands, service discovery may be used as shown in Figure 4.2.

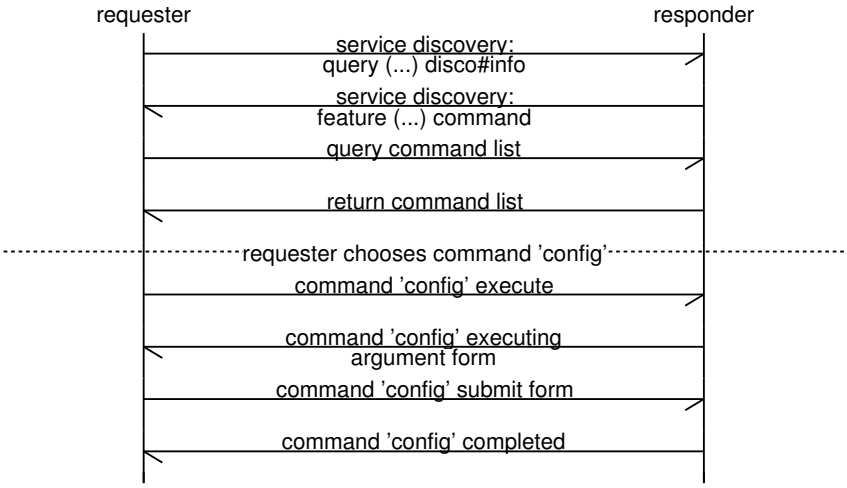

Figure 4.2: data flow in adhoc command discovery and execution

Simple command execution – for example using commands that do not require any arguments – the requester chooses a command node and sends the execute request to the responder (the client that executes the request) which in turn answers with a status `completed`.

Commands that require arguments need to make use of XMPP forms as specified in XEP-0004 [101] that allow exchanging such data in a similar fashion to how HTML forms work. The responder then responds with the form to be filled out by the requester:

```
<iq><command xmlns='http://jabber.org/protocol/commands' ...>
  <x xmlns='jabber:x:data' type='form'>
    <title>choose server to ping</title>
    <field var='server' label='Server' type='list-single'>
      <option><value>web.example.com</value></option>
      <option><value>mail.example.com</value></option>
      <option><value>dns.example.com</value></option>
```

```
    </field>
  </x>
</command></iq>
```

The requester then fills out the form and sends it to the responder in order to continue command execution. Adhoc commands are not limited to one iteration of form exchange but are quite flexible in how commands are structured and organized. Mechanisms to move forward and backward in the different states are available as actions: `next`, `prev` and `complete`.

Access control to commands – which JID is allowed to run which commands – is up to the client implementation and configuration. This is in line with what XEP-0050 suggests that defines `<xmpp:forbidden/>` as the error message for these cases.

During any step of the command execution, a party may issue a `cancel` to stop the command execution. Additional information about the cause of the cancelation may be given in a `<note />` entity.

Compared to ordinary XMPP messages, adhoc commands offer several advantages:

- The use of service discovery for discovering the abilities of other parties in the network.
- Structured data exchange through the use of forms.
- Completeness check of the the data and simple type conversions by using elementary data types like `boolean`, `text-single` or `list-single` to name a few that are alread defined in XEP-0004.
- Multi stage command execution with automatic tracking of the command session.
- Error handling mechanisms.

Adhoc commands provide a powerful way to safely execute commands on a remote client provided that one uses the already present features as a safety guard: every command needs a single purpose, a selected list of arguments and an access control to avoid arbitrary command execution. A command like `RunShellCommand` with a free-text field for the shell command and arguments is the wrong way to go. At first this might sound like a good idea and there even exist some simple command bots that offer just that (see Section 4.4 about such bots). On the other hand, it is exactly this approach that makes, for example, NRPE a doublesided sword.

Predefining the really needed commands in day-to-day system monitoring and management and implementing adhoc commands helping to fullfil just these needs brings a powerful tool.

Server Components

A very powerful mechanism to extend the features of XMPP servers are components that are directly connected to the XMPP server using a protocol that is defined in XEP-0114 [173]. Server components usually get all traffic forwarded destined for a subdomain they are responsible for. The XMPP server itself is only used for message routing and the server component may only use the XMPP server as a message relay responsible for stanza delivery. All Roster management and other means of access control need to be implemented in the component itself.

The aforementioned PubSub service (pubsub.example.com) is implemented as a server component on some XMPP servers. As components have no access to server internals and thus the user's roster, they cannot implement certain access models defined in XEP-0060 like the one based on a roster group (`roster`)or the users subscription status (`presence`).

Another example of a server component is the Multi User Chat (MUC) – usually located at conference.example.com org conf.example.com – that is specified in XEP-0045 [172] and superseeds the original Groupchat-1.0. The functionality of MUC is clearly to mimik the concept of IRC chatrooms. The component itself needs to assign a JID to each chatroom. For example, a room with the name "test" can be reached with the JID `test@conference.example.com`. Associations between users and rooms, presence handling and message forwarding has to be done by the server component.

4.4 Management and Monitoring Systems based on XMPP

Back when XMPP was initially released to a wider audience as Jabber, many system administrators started exploring the new protocol by porting – or rewriting – some of the predominant IRC bots used in system administration. Many of these smaller tools are still in use but also more complete systems have been written that also make use of the more advanced features of the XMPP protocol.

Kestrel

The university of Clemson operates a High Performance Computing (HPC) cluster named Palmetto [49] currently consisting of more than 20.000 cores. Distribution and management of jobs on clusters of such a scale provides many challenges. Lance Stout and others wrote Kestrel [176], a job manager for HPC clusters. Kestrel uses XMPP at its core for job management.

At its core, the Kestrel manager is implemented as a server component, mainly due to scalability issues with the manager's roster: a huge roster with several hundred items or more significantly slows down XMPP communication, especially on startup when the roster as a whole needs to be transferred to the client (the Kestrel manager in this case). A server component offers the advantage that all roster management needs to be done by the component itself, thus removing the bottleneck.

Kestrel defines its own messages by defining its own namespaces `kestrel:job` or `kestrel:tasks`. Using these namespaces allows the use of service discovery and thus again use one of the XMPP core features to get parts of the work done. These stanzas taken from Lance Stout's thesis about the creation of Kestrel [176] illustrate how such communication may look like:

```
<iq to="worker21@example.org" type="get">
  <query xmlns="http://jabber.org/protocol/disco#info"
         node="kestrel:tasks:capabilities" />
</iq>
<iq from="worker21@example.org" type="result">
  <query xmlns="http://jabber.org/protocol/disco#info">
```

```
    <feature>Python2.6</feature>
    <feature>Linux</feature>
  </query>
</iq>
```

A job submission taken from the same publication may look like this:

```
<iq type="set" to="manager.example.org">
  <job xmlns="kestrel:job" action="submit" queue="50000">
    <command>/runtask.sh</command>
    <cleanup>/cleanfiles.sh</cleanup>
    <requires>Python2.6</requires>
    <requires>SleekXMPP</requires>
  </job>
</iq>
```

Based on the features, workers provide and report via service discovery after they initially connected to the master, the master then may select workers providing all the required features for job distribution.

Kestrel does not do data or dataset distribution on its own but relies on external tools like for example web servers or shared storage to take care of this. Federation and clustering allow to even further scale Kestrel. For typical cluster workloads, Kestrel offers high performance and low latency and the approach taken by its developers is well suited.

Archipel Project

Archipel [4] is a real time orchestration tool for virtualization clusters that is built from several parts: a web interface that directly interacts with the XMPP server via web sockets, an agent that runs on every hypervisor managing the virtual machines and optionally a central agent that allows management of virtual machines over several hypervisors.

The project recommends to use a separate XMPP server for use with Archipel, because of several special settings regarding maximum stanza sizes, heavily lifted bandwidth limits and the need for auto-registration being enabled for creating XMPP accounts for new virtual machines. Those settings are necessary to allow real-time visualization of virtual machine and hypervisor states. A central mechanism for communication is realized through PubSub nodes on the XMPP server, the some of the nodes are configured to store items in a persistent way: `/archipel/roles` for example is used to store permission templates.

Archipel also uses its own namespaces for service discovery and efficient communication. `archipel:hypervisor` denotes commands and information regarding the hypervisor whereas `archipel:vm` refers to virtual machines. An agent will have one XMPP account being connected for the hypervisor and one for each virtual machine. An administrator may delegate permissions to manage hypervisors or virtual machines to other users. Through S2S, federation is possible.

Smaller Tools

XMPPMote [178] is a simple command daemon that allows command execution on a server with full administrative privileges. Unfortunately this bot does not implement any kind of roster management or presence subscription support: before it is possible to communicate with the bot a presence subscription between the bot's account and those of the people allowed to talk to the bot needs to be established with a different XMPP client.

SyslogBot [180] is a simple daemon that forwards messages from a named pipe to a defined list of JIDs. Syslog daemons like syslog-ng or rsyslog may be configured to output messages to a named pipe. The bot does not implement any kind of access control but accepts to exchange messages with anyone talking to it. As the bot is mainly forwarding local syslog messages to configured JIDs, only the other commands to get memory usage, system load or running processes disclose information to anyone.

Pysysbot [68] provides access to information about system time, uptime, current load, number of running processes, disk and memory usage. This bot – exactly like SyslogBot – does not provide any means of access control and will exchange messages with anyone knowing the bot's JID.

There are many more simple XMPP bots using chat messages for communication and lacking basic security methods. Basically the security level of these bots can be considered similar to those of classic IRC bots.

Common Problems and Misconceptions

XMPP as a protocol is well suited for utilization in applications that are way beyond instant messaging. Besides being completely push-based, the protocol offers message queue like features and generally scales very well, but there are also weaknesses in the protocol: mainly transferring of larger amounts of data is slow, and even artificially slowed down to evenly distribute bandwidth between all users. XMPP servers limit the maximum stanza size to 64KB by default and the maximum bandwith per connection to 1KB per second.

Applications like the above mentioned Archipel require the limits in the XMPP server to be raised considerably to allow exchaning larger amounts of data to come over that limitation. A south african Internet Service Provider (ISP) used to use XMPP extensively for provisioning [127], but they gave up on it and switched to a plain message queuing tool because they couldn't cope with roster management.

Roster Management

In environments where XMPP is used to connect bots to each other, the use of a roster is not strictly necessary. When rosters grow too big – when they reach a size beyond 64KB – startup and connection initialization becomes slow. Mechanisms to avoid big rosters have been introduced in the course of this chapter: PubSub and server components.

Another adverse effect of a huge roster is the amount of presence data a bot receives: the server delivers presence updates for any account on the bot's roster. XMPP libraries like Engineyard's Vertebra [124] automatically adds anyone requesting a subscription to the roster.

Jack Moffitt, an enterpreneur and founder of Chesspark, an online chess playing platform running on XMPP chess bots, published [148] some interesting thoughts about scaling XMPP bots even further: by using the S2S protocol and mimic a server.

When using XMPP in an automated environment, one has to deal with roster management and think about how to handle rosters in advance.

File Transfers

Several XEPs specify file transfer with XMPP. While most of these proposals try to avoid using XMPP directly as a transport (in-band) and suggest the use of other means of transport like proxy servers (out-of-band). XEP-0047 [132] defines In-Band Bytestream (IBB) transport that is used as the last resort for file exchange when no other methods can be agreed on by both parties.

While out-of-band file transfers usually have advantages in transfer time, they lack the protection of the encrypted and authenticated connection, XMPP offers. When using XMPP for file transfers, this may work out for small files using IBB but for bigger amounts of data, other means of transport need to be used. With out-of-band transfer methods special care has to be taken to ensure confidentiality, integrity and authenticity.

Other Concepts

As we have seen, Archipel as well as Kestrel use their own namespace, thus integrate their semantics into XMPP. This offers the huge advantage that one doesn't need to encapsulate a different application level protocol and use XMPP as a transport protocol only. Using an own namespace improves parsing and message handling and is a step towards tighter integration into XMPP.

The use of separate namespaces also has the advantage that service discovery may be used to find other entities providing services.

"Every new hire is a potentially unpatched set of vulnerabilities."

Adam Baldwin, Chief Security Officer at &yet, March 2014

CHAPTER 5

Implementation

5.1 Prerequisits

XMPP Server

The XMPP server used throughout this section is ejabberd [20] v15.06, although other versions beginning with v14.05 can be used too[1]. `servers.example.com` is the XMPP domain name with `servername@servers.example.com` being used for the accounts (unique server names within a network are assumed) which were pregenerated by the system's administrators.

Default values are used for the server settings except for the shaper where the client limit was raised to 5000 bytes per second (from a default of 1000). This value allows for faster in-band file or data transfer rates which the server shapes to 5000 bytes per second. The maximum stanza size, however, is limited to the default of 65536 bytes. Note that those settings will suffice for common networks but may lead to data arriving slowly for hosts collecting or sending data at higher rates: transfering a stanza of 64KB will take more than a minute using the default shaper while using a limitation of 5000 bytes per second will reduce the transfer time to about 14 seconds. Depending on the average data sizes the limit may be rised even further for the XMPP server hosting `servers.example.com`.

XMPP Library and Language Choice

The language used for prototype implementation is Python [51], a high-level object-oriented language suitable for rapid prototyping that provides many libraries either within the language's

[1] older versions of ejabberd had limitations in their cipher settings especially with regard to Diffie-Hellman parameters that allowed weak cryptography to be used on the wire. Their use is therefore not recommended. For details on state-of-the-art crypto settings please see bettercrypt.org's [6] recommendations in general and RFC7590 [174] and xmpp.net [67] for specific recommendations on XMPP security

core or as addons to interface with many different protocols. For XMPP protocol support SleekXMPP [58], a library that offers support for most XEPs and is multithreaded at its core, was chosen. Both – Python and SleekXMPP – have been used in the development of Kestrel [176] too and are well supported on different Linux platforms.

A few very recent features require the use of SleekXMPP in version 1.3.1 or newer – these include the new Internet of Things (IoT) XEPs like XEP-0323 [187] or XEP-0325 [185]. All other features are built with the intention to work with version 1.0-beta5 onwards because that version is supported on most currently used Linux platforms and can be installed from the trusted software repositories of the Linux distributions in use.

5.2 Basic Design

An agent runs on every server, participating in the XMPP network via an account per server. The configuration file provides a list of administrators and server features the agent needs to communicate with; based on that data, the agent manages its Roster: it first retrieves the current Roster from the server, removes all items that are not listed in the configuration file and then adds those that previously were not listed there but are in the configuration.

Configuration

Configuration is currently only handled via file, configuration updates over the network are not implemented as this would require a very detailed permissions model throughout the whole agent to still allow the system on which an agent runs on to stay autonomous and just delegate certain tasks and information to more centralized components. The current permissions model allows every plugin to specify an Access Control List (ACL) containing a list of JIDs allowed to access its features.

As the agent is heavily multithreaded, the configuration is implemented as a Singleton which is achieved by on-the-fly creation of a module: in Python – the language of choice for the prototype – a module is by definition a Singleton and internally handled as such. Even some core features like the Python `logging` module [52] uses this property of a module in python. The configuration file itself uses ini-style which means that the configuration consists of sections that hold key-value pairs containing configuration information.

```
[main]
socketfile: /var/run/ypke/socket
pidfile: /var/run/ypke/pid
logfile: /var/log/ypke.log
loglevel: DEBUG

[xmpp]
jid: aserver@servers.example.com
password: s3cr3t
acl: user@example.com
```

Core

The core system of an agent provides three distinct interfaces to interact with:

- a **scheduler** for routine tasks
- a **socket** allowing local services to interact with the agent.
- an **XMPP** interface for incoming and outgoing communication.

The multithreaded and event driven nature of SleekXMPP inspired the implementation of the agent too: all actions and events take place in threads, the agent is non-blocking. As the Roster is managed based on the configuration file, the socket interface implements similar access control based on user IDs.

The core of the agent just provides the infrastructure, all features need to be implemented in plugins.

Plugins

On startup, the agent reads the configuration and loads all plugins configured there. Plugins provide all their functionality using some of the features the agent core provides by implementing one or more of:

- a routine task
- a command called via socket
- a command called via XMPP
- sending a message via XMPP
- an adhoc command
- picking up data via PubSub

Plugins need to be specific in what they allow to do: a generic "Run any command with administrative privileges" is considered harmful and therefor, of course, not provided.

On a more technical level, for a plugin to be loadable by the agent, it needs to provide a `register()` function that returns a tuple of lists containing the features it implements for sockets, xmpp and routine tasks. Plugins already have the whole infrastructure of the agent available for their use: they may access the configuration, use XMPP in the scope of the server running the agent and may access other agents.

5.3 Legacy Support

To be able to interact with currently available means of data exchange some legacy interfaces to collect data were implemented: this includes SMTP as discussed in Section 4.1, Syslog as discussed in Section 3.3 and the Simple Network Management Protocol (SNMP) discussed in Section 3.4 used to collect counter information from network equipement.

Sendmail Interface

Besides being the name of a common UNIX mail server [55], `sendmail` is the name of a command that can be expected to be available on every Linux server with the purpose of sending email messages. The consumer of the command does not need to know any details about the mail system configuration and may just use that command to get an email delivered to its recipient.

The Linux Standard Base (LSB) provides a detailed description about the commandline switches, the command has to support [32]. Implementing this interface allows to take over delivery of email messages generated on a server.

The implementation consists of two parts: the `sendmail` binary itself providing all the necessary interfaces required by the LSB and an agent plugin that picks up email messages via the socket interface and forwards them – depending on the configuration – via either XMPP or SMTP or both[2].

`sendmail` Implementation

The `sendmail` command implements all commandline switches required by the LSB standard. These mainly require two modes of operation:

- `-bm` – read message from `stdin` and deliver it to the recipients specified either within the message (requires `-t` to be specified too) or given as an argument.
- `-bs` – communicate via SMTP on `stdin` with the caller to retrieve the message. All recipients are specified within the SMTP session.

An email message recieved via either of the two modes is passed to the agent on the socket file where the agent returns the message id after successful queueing.

Plugin Implementation

The xmppmail plugin implements a socket command to receive messages from the `sendmail` command, manages its own queues on disk to avoid losing messages (even on server crashes or sudden reboots) and forwards email messages according to the configuration.

```
[plugin.xmppmail]
spool: /var/spool/xmppmail

# smtp config
smtp: true
smtpserver: mail.example.com
smtptls: true
smtpauth: false
#smtpuser:
```

[2] a standalone implementation named xmppmail is available too to compensate for some of the weaknesses in current mail-forwarding solutions like nullmailer discussed in 4.1

```
#smtppass:
#smtpfrom:
smtpto: admin@example.com

# xmpp config
xmpp: true
xmppto: admin@example.com, user@example.com
```

The implementation tries to mimic that of a Nullmailer [45] setup: emails are being accepted from local applications via `sendmail` and forwarded to a pre-configured email address via a default SMTP server. XMPPMail does not try to be a fully fledged SMTP server, but tries to provide several of the features that Nullmailer has to forward mails to a central mail server in a secure and reliable way:

- force the use of encryption (`smtptls: true`) or refuse to deliver an email in case an encrypted connection cannot be established.
- allow to authenticate with username and password at the remote host (a feature that Nullmailer gained in version 1.03, six years before implementing transport security to actually protect those credentials on the wire with version 1.10).

In the context of the agent, other plugins that deal with event generation may be used to analyze the email before it is being forwarded and possibly trigger certain actions related to that email besides the XMPPMail plugin forwarding the message.

During the development of this plugin, it turned out that a simple communication mechanism is sufficient for use with the socket: every client sends the command it wants to use as the first space separated argument; the agent uses that to determine the plugin responsible for handling the conversation and hands over the socket to that plugin.

The plugin provides a pattern database that helps with classification of a message and may be extended by the user allowing to parse email messages and match them into an event. While sending messages via SMTP isn't changed in any way, XMPP delivery is influenced by the classification of a message: a message not matched by any of the patterns in the database will immediately be sent to the administrators as an alert. All other messages will be treated according to their classification in the patterns database and if they are informational only, the won't be sent to the administrators.

Syslog Collector

To collect syslog data of a host, rsyslog is used and extended in a way that syslog messages are fed into a plugin for further processing via a named pipe. The plugin then uses Marcus Ranum's approach of "Artificial Ignorance" with the help of logcheck to filter out possibly important syslog messages. A simple additional parser with patterns parses the remaining messages and classifies them. Messages not recognized by the parser patterns are immediately sent to the system administrators as an alert. The administrators then should classify the message and add it to the pattern collection of the plugin. This helps ensuring that even less false alarms will be raised once enough patterns have been seen and analyzed.

SNMP Collector

Collecting data via SNMP from switches and others is implemented by a plugin; the data is either stored in RRD files locally or – if configured – sent to a server component, by default `stats.servers.example.com`, as RRD data series that the component then adds to a RRD file.

The component needs to know which JID is responsible for delivering data for which switches in order to accept data submitted. The collector plugin submits the data to a JID reflecting the name of the switch: `coreswitch@stats.servers.example.com`.

5.4 Management Support

Basic commands implemented in plugins should always define their own namespaces to allow service discovery and to minimize the need for prior knowledge about the systems. It is of course possible to create a plugin that acts upon simple messages containing commands – as the smaller bots in Section 4.4 do – but that has the disadvantage of the absence of service discovery and is thus to be avoided.

A basic plugin that allows rebooting and shutting down a server is provided; more advanced commands like handling updates are not implemented at the moment.

Distributed Commands

Some tasks in network management consist of several tasks that have to be executed on different servers: creating a new user's account is probably one of the most famous examples fitting in this category. Usual steps to accomplish this include:

- create an account in a central authentication database
- create the user's home on a data server
- create a mailbox and some mail aliases on a mail server
- add the user to the staff page on the web server
- create a public_html for the user's personal web page.
- (and maybe many more tasks on different servers)

Some of the tasks have to be done before others, like the creation of the account which then provides user and group id for directory creations. The common approach to these kinds of tasks involve a series of scripts that an administrator needs to execute on a series of servers. A notification about a new staff member sent by the front office triggers the administrator to run these scripts and finally hand over a document containing the most important information about the newly created account to the new staff member.

Delegation

Allowing the front office to create the user account by themselves would remove the pressure from the administrative team to immediately act but is in most cases impossible due to the fact that most of these scripts require the highest privileges (`root`) a user may have on a system. Using adhoc commands as specified in XEP-0050 [146] on an agent via XMPP could make it possible to delegate privileged actions to normal users.

Creation of user accounts is certainly not part of the daily business of a small to medium sized organization. Therefor it is advised to implement a master agent that triggers all the single steps required to create an account. This helps in minimizing possible mistakes and inconsistencies. Although the creation of a new account is part of any IT infrastructure maintainance, there is no general way to "do it": from the above assumptions, we may deduce four distinct roles involved in account creation:

- `master` – create the account, then trigger the three other roles to act.
- `fileserver` – create user home.
- `mailserver` – create mailbox and aliases.
- `webserver` – add to staff page, create personal web space.

Within the plugin those roles are implemented with the `master` role being the one that may be triggered; the other roles are just allowed to be run by the master server. On an organizational level the hiding of the complexity of an account creation by providing a single command that can be run by an end user makes that irregular task easier handle because single steps in the process cannot be missed.

```
[plugin.create_user]
role: master
acl: [admin@example.com, office@example.com]
```

Command Execution

Adhoc commands rely on forms (as described in XEP-0004 [101]); after using service discovery to actually ensure the other agent supports a certain adhoc command, the initiator starts the command execution process by sending an `iq` stanza with the action being `execute` to the remote agent. That agent responds with an `iq` stanza containing the form with all the arguments required to execute the command (see Code Listing 1 for a function generating such a form) and a status of `executing` indicating that the command execution is already in progress.

The initiator then fills out the required form fields and sends the form back in an `iq` stanza of the type `set`. The recieving agent then validates the form data and finally proceeds with the actual execution of the command after which it returns the result with a status of `completed` (or an error message in case an error happened).

The execution of a command can be triggered either with the help of a XMPP client that supports adhoc commands or through a web interface. That way, complex administrative commands

```
def start_create_user(iq, session):
    account_form = xmpp.make_form('form', 'create_user')
    account_form.addField(var = 'login',
                          ftype = 'text',
                          label = 'account name')
    (...)
    session['payload'] = account_form
    session['next'] = create_user
    return session
```

Code Listing 1: function `start_create_user` that prepares the form and sends it back to the initiator

may be delegated to unprivileged user accounts without having to fear elevated priviledges may endanger the integrity of the servers involved.

5.5 Data Collection

Systems collect performance data and statistics locally with sysstat and, if available, ambient temperature and voltage data with the help of builtin sensors. Some of these values, mainly system load and temperature data, are closely monitored on the systems themselves and alerts are triggered in case values exceed their thresholds. For more detailed performance analysis, a history of that data is kept on the system and may be inspected there.

With the help of a plugin other entities may request certain performance data. The plugin then creates a PubSub node, subscribes the entity to that node and publishes the requested performance data to that node. The other entity may then use that data; at the moment, only storing that data to a rrd file is implemented so that graphing the data is possible too. It is possible but not implemented to feed that data into an interactive web interface in a way similar to the performance monitoring Archipel does.

The plugin provides an interface to request data and to cancel a subscription for data and handles PubSub node creation and deletion and the publishing of the requested data. A planned enhancement is to allow changeing the data aquisition interval which is 10 minutes at the moment.

The above mentioned server component that collects SNMP statistics from switches may also subscribe to performance data of systems.

5.6 Event Collection

Many of the plugins and parts of this framework create events that are not relevant to system administrator like, for example, failed logins due to malicious account probing. Such information may be relevant for other components on a network or even other network owners.

A server component, `events.servers.example.com`, has been implemented that collects and aggregates such information. When three failed logins from an IP address have

been recorded with an hour – no matter on which system – the component publishes that ip with a reasoning like "login probing" to a PubSub node that any server may subscribe to. The servers then can decide on their own whether they want to block that IP address with a firewall rule and for how long they want to block that IP and so on.

Such information about login probing or other generic incidents like web scanners trying to automatically find vulnerable web applications are even relevant for a wider audience and may be shared with others.

5.7 Contribution

A new approach to integrate and use SMTP in a monitoring system has been shown: with the implementation of the sendmail POSIX interface, mail notifications of server systems can easily be picked up, analyzed and used in a monitoring system.

Although many XMPP bots for system management or monitoring are available, they only use messaging features of the XMPP protocol and completely ignore the more advanced features of the protocol that add value and allow a monitoring system to scale. Those features include service discovery, the use of own namespaces that allow providing APIs other parts of such a system may rely on, PubSub for completely decentralized communication aspects and server components for more centralized parts of a monitoring infrastructure like the collection of statistics or the analysis of events in a network.

5.8 Future Work

Using the system introduced as it is now, federation with other XMPP servers imposes trust issues: an administrator of another server may take over an account on his server that is allowed to manage some servers. So delegation to JIDs of other hosts isn't recommended at the moment. Future enhancements should therefor include mechanisms like the one introduced in XEP-0027 [149] that describes how to use Pretty Good Privacy (PGP) over XMPP. With addition of PGP, repudiation and confidentiality may be added even when the XMPP servers aren't self controlled.

Pushing the idea of decentralized management a little further, use of serverless XMPP may be explored to let systems organize in a rather autonomous way. To make that possible, PGP needs to be implementented and alternative mechanisms for PubSub need to be found.

With the advent of PGP, agent configuration can be done in-band more easily: keeping an archive of signed configuration change requests even provides an audit history. Mechanisms to distribute parts of a configuration either via PubSub or with the help of Adhoc commands is imaginable.

To facilitate initial deployment, agent servers may register their account at the XMPP server while the administrator then only has to acknowledge the account. Having such a mechanism at hand for the initial software install, deployment suddenly gets very easy: just installing the agent software suffices as this will trigger account registration. Together with in-band configuration management, deployment of agents is easy.

In the process of event collection, an important feature still missing is the expectation of job completion within a certain time frame. Backup jobs started by the system's periodic scheduler

sending an email on completion are an example of such jobs: to make sure, backup ran – and even completed successfully – a scheduler expecting events to happen before a deadline needs to be implemented.

The system still has its rough edges but a solid foundation for future enhancements has been made. By using the XMPP protocol beyond simple chat messages, its qualification as the heart of a monitoring and management system has been shown.

List of Figures

List of Tables

Bibliography

[1] Aim instant messenger. http://www.aim.com/.

[2] Apache wave homepage. http://incubator.apache.org/wave/.

[3] Apple push notification service. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conce

[4] Archipel project homepage. http://www.archipelproject.org/.

[5] Artificial ingorance. published on the firewall-wizards list.

[6] Bettercrypto.org homepage. https://www.bettercrypto.org/.

[7] Brubeck – a statsd implementation for github. http://githubengineering.com/brubeck/.

[8] Brubeck implementation. https://github.com/github/brubeck.

[9] check_mk business intelligence. http://mathias-kettner.de/checkmk_bi.html.

[10] check_mk configuration & check engine. http://mathias-kettner.de/checkmk_ways_to_install.html.

[11] check_mk event console. http://mathias-kettner.de/checkmk_mkeventd.html.

[12] check_mk homepage. http://mathias-kettner.de/check_mk.html.

[13] check_mk livestatus. http://mathias-kettner.de/checkmk_livestatus.html.

[14] check_mk micro core. http://mathias-kettner.de/cms_cmc.html.

[15] check_mk multisite. http://mathias-kettner.de/checkmk_multisite.html.

[16] check_mk security considerations. http://mathias-kettner.de/checkmk_security.html.

[17] collectd – the system statistics collection daemon. https://collectd.org/.

[18] Counting and timing. http://code.flickr.net/2008/10/27/counting-timing/.

[19] Cve details – nagios : Security vulnerabilities. https://www.cvedetails.com/vulnerability-list/vendor_id-1424/Nagios.html.

[20] ejabberd homepage. http://www.ejabberd.im/.

[21] Elasticsearch homepage. http://elasticsearch.org/.

[22] Etsy statsd implementation. https://github.com/etsy/statsd.

[23] Facebook chat api. https://developers.facebook.com/docs/chat/.

[24] Ganglia homepage. http://www.ganglia.info/.

[25] Google cloud messaging. https://developers.google.com/cloud-messaging/.

[26] Google talk. https://developers.google.com/talk/open_communications?csw=1.

[27] Heise security: Whatsapp stopft sicherheitsloch. http://heise.de/-1753088.

[28] History of IRC (internet relay chat). http://daniel.haxx.se/irchistory.html.

[29] Icinga homepage. https://www.icinga.org/.

[30] Icq instant messaging. http://www.icq.com/.

[31] Liblognorm homepage. http://www.liblognorm.com/.

[32] Linux standard base – sendmail interface. http://refspecs.linuxbase.org/LSB_5.0.0/LSB-Core-generic/LSB-Core-generic/baselib-sendmail-1.html.

[33] List of statsd implementations. http://www.joemiller.me/2011/09/21/list-of-statsd-server-implementations/.

[34] Logcheck homepage. http://www.logcheck.org/.

[35] Logstash homepage. http://www.logstash.net/.

[36] Mitre: Common event expression homepage. https://cee.mitre.org/.

[37] The monitoring plugins project. https://www.monitoring-plugins.org/.

[38] Naemon homepage. http://www.naemon.org/.

[39] Nagios and netsain history. https://www.nagios.org/about/history/.

[40] Nagios homepage. https://www.nagios.org/.

[41] Nagios ndoutils. https://exchange.nagios.org/directory/Addons/Database-Backends/NDOUtils/details.

[42] Nagois plugin – check_by_ssh_master. https://exchange.nagios.org/directory/Plugins/*-Remote-Check-Tunneling/check_by_ssh_master/details.

[43] Nrpe - nagios remote plugin executor. http://nrpe.sourceforge.net/.

[44] Nsca - nagios service check acceptor. https://exchange.nagios.org/directory/Addons/Passive-Checks/NSCA–2D-Nagios-Service-Check-Acceptor/details.

[45] Nullmailer homepage. http://untroubled.org/nullmailer/.

[46] Open source monitoring conference 2013. https://www.netways.de/en/events_trainings/osmc/archive/osmc2013/.

[47] Openwrt – wireless freedom. https://openwrt.org/.

[48] Original statsd implementation. https://github.com/iamcal/Flickr-StatsD.

[49] Palmetto: Clemson university's high performance computing cluster. http://citi.clemson.edu/palmetto/.

[50] Project lumberjack homepage. https://fedorahosted.org/lumberjack/.

[51] Python homepage. http://www.python.org/.

[52] Python logging module. https://docs.python.org/2/howto/logging.html.

[53] Rsyslog is the rocket-fast system for log processing. http://www.rsyslog.com/.

[54] Sap announcing google wave support. https://scn.sap.com/people/daniel.graversen/blog/2009/08/21/sap-enterprise-service-and-google-wave.

[55] Sendmail homepage. http://www.sendmail.com/.

[56] Shinken homepage. http://www.shinken-monitoring.org/.

[57] Skype homepage. http://www.skype.com/.

[58] Sleekxmpp homepage. http://www.sleekxmpp.com/.

[59] Statsd specification. https://github.com/b/statsd_spec.

[60] Supervisord homepage. http://supervisord.org/.

[61] syslog-ng: Open source log management solution. https://www.syslog-ng.org/.

[62] Systemd homepage. http://www.freedesktop.org/wiki/Software/systemd.

[63] UWSGI homepage. http://projects.unbit.it/uwsgi/.

[64] Whatsapp? nicht ohne risiken. http://shakal.blog.de/2011/03/22/whatsapp-risiken-10872342/.

[65] Whatsapp protocol - funxmpp. https://github.com/WHAnonymous/Chat-API/wiki/FunXMPP-Protocol.

[66] The whisper database. http://graphite.readthedocs.org/en/latest/whisper.html.

[67] Xmpp.net homepage. http://www.xmpp.net/.

[68] Fabian Affolter. pysysbot: A simple python jabber bot for getting system information. https://github.com/fabaff/pysysbot.

[69] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), March 2005. Updated by RFCs 6014, 6840.

[70] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014, 6840.

[71] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014, 6840, 6944.

[72] Joshua Barratt. Signal from noise. Scale9x, 2011.

[73] U. Blumenthal, F. Maino, and K. McCloghrie. The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model. RFC 3826 (Proposed Standard), June 2004.

[74] U. Blumenthal and B. Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). RFC 3414 (INTERNET STANDARD), December 2002. Updated by RFC 5590.

[75] Mark Burgess. Linux journal: Promise theory – what it is. http://www.linuxjournal.com/content/promise-theory

[76] Mark Burgess and Siri Fagernes. Promise theory-a model of autonomous objects for pervasive computing and swarms. In *null*, page 118. IEEE, 2006.

[77] J. Case, D. Harrington, R. Presuhn, and B. Wijnen. Message Processing and Dispatching for the Simple Network Management Protocol (SNMP). RFC 3412 (INTERNET STANDARD), December 2002. Updated by RFC 5590.

[78] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Coexistence between version 1 and version 2 of the Internet-standard Network Management Framework. RFC 1452 (Proposed Standard), April 1993. Obsoleted by RFC 1908.

[79] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Conformance Statements for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1444 (Proposed Standard), April 1993. Obsoleted by RFC 1904.

[80] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Introduction to version 2 of the Internet-standard Network Management Framework. RFC 1441 (Historic), April 1993.

[81] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1450 (Proposed Standard), April 1993. Obsoleted by RFC 1907.

[82] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Manager-to-Manager Management Information Base. RFC 1451 (Historic), April 1993.

[83] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1448 (Proposed Standard), April 1993. Obsoleted by RFC 1905.

[84] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1442 (Proposed Standard), April 1993. Obsoleted by RFC 1902.

[85] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1443 (Proposed Standard), April 1993. Obsoleted by RFC 1903.

[86] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Transport Mappings for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1449 (Proposed Standard), April 1993. Obsoleted by RFC 1906.

[87] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework. RFC 1908 (Draft Standard), January 1996. Obsoleted by RFC 2576.

[88] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1904 (Draft Standard), January 1996. Obsoleted by RFC 2580.

[89] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Introduction to Community-based SNMPv2. RFC 1901 (Historic), January 1996.

[90] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1907 (Draft Standard), January 1996. Obsoleted by RFC 3418.

[91] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1905 (Draft Standard), January 1996. Obsoleted by RFC 3416.

[92] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1902 (Draft Standard), January 1996. Obsoleted by RFC 2578.

[93] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1903 (Draft Standard), January 1996. Obsoleted by RFC 2579.

[94] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1906 (Draft Standard), January 1996. Obsoleted by RFC 3417.

[95] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction to Version 3 of the Internet-standard Network Management Framework. RFC 2570 (Informational), April 1999. Obsoleted by RFC 3410.

[96] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and Applicability Statements for Internet-Standard Management Framework. RFC 3410 (Informational), December 2002.

[97] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol. RFC 1067, August 1988. Obsoleted by RFC 1098.

[98] J. Davin, J.D. Case, M. Fedor, and M.L. Schoffstall. Simple Gateway Monitoring Protocol. RFC 1028 (Historic), November 1987.

[99] M. Dilman and D. Raz. Efficient reactive monitoring. *Selected Areas in Communications, IEEE Journal on*, 20(4):668–676, May 2002.

[100] R. Droms. Dynamic Host Configuration Protocol. RFC 2131 (Draft Standard), March 1997. Updated by RFCs 3396, 4361, 5494, 6842.

[101] R. Eatmon, J. Hildebrand, J. Miller, T. Muldowney, and P. Saint-Andre. Xep-0004: Data forms. XMPP Standard Proposal, August 2007.

[102] Eugene Eberbach. Challenges facing computer science in the 21st century, 2001.

[103] Ronald Eikenberg. Drupal lücke mit dramatischen folgen, October 2014. [Online; posted 2014-10-29].

[104] Kensuke Fukuda. On the use of weighted syslog time series for anomaly detection. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 393–398. IEEE, 2011.

[105] Jean Gabès. New nagios implementation proposal. http://sourceforge.net/p/nagios/mailman/message/24087464/.

[106] Ethan Galstad. Nagios – a fork in the road. http://community.nagios.org/2009/05/11/nagios-a-fork-in-the-road/.

[107] J. Galvin and K. McCloghrie. Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1445 (Historic), April 1993.

[108] J. Galvin and K. McCloghrie. Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1446 (Historic), April 1993.

[109] R. Gerhards. The Syslog Protocol. RFC 5424 (Proposed Standard), March 2009.

[110] Sebastien Godard. Sysstat homepage. http://sebastien.godard.pagesperso-orange.fr/.

[111] G. Goldszmidt and Y. Yemini. Distributed management by delegation. In *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*, pages 333–340, May 1995.

[112] Dawid Golunski. Exploitdb: Nrpe <= 2.15 - remote command execution. https://www.exploit-db.com/exploits/32925/.

[113] W. Hardaker. Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP). RFC 6353 (INTERNET STANDARD), July 2011.

[114] D. Harrington and W. Hardaker. Transport Security Model for the Simple Network Management Protocol (SNMP). RFC 5591 (INTERNET STANDARD), June 2009.

[115] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411 (INTERNET STANDARD), December 2002. Updated by RFCs 5343, 5590.

[116] D. Harrington, J. Salowey, and W. Hardaker. Secure Shell Transport Model for the Simple Network Management Protocol (SNMP). RFC 5592 (Proposed Standard), June 2009.

[117] D. Harrington and J. Schoenwaelder. Transport Subsystem for the Simple Network Management Protocol (SNMP). RFC 5590 (INTERNET STANDARD), June 2009.

[118] R. Hartmann. Default Port for Internet Relay Chat (IRC) via TLS/SSL. RFC 7194 (Informational), August 2014.

[119] J. Hildebrand, P. Millard, R. Eatmon, and P. Saint-Andre. Xep-0030: Service discovery. XMPP Standard Proposal, June 2008.

[120] Wilco Baan Hofman. Nagios bugtracker – bug #90. http://tracker.nagios.org/view.php?id=90.

[121] IEC. *ISO/IEC 8824 (1987): Information processing systems — Open Systems Interconnection — Abstract Syntax Notation One (ASN.1)*. 1987.

[122] IEC. *ISO/IEC 7498-4 (1989): Information processing systems — Open Systems Interconnection — Basic Reference Model – Part 2: Security Architecture*. 1989.

[123] IEC. *ISO/IEC 7498-4 (1989): Information processing systems — Open Systems Interconnection — Basic Reference Model – Part 4: Management framework*. 1989.

[124] Engine Yard Inc. Vertebra. https://github.com/engineyard/vertebra.

[125] P. Hancke J. Miller, P. Saint-Andre. Xep-0220: Server dialback. XMPP Standard Proposal, April 2014.

[126] Dave Josephsen. Changing the game, part 3.

[127] Kenneth Kalmer. To amqp or to xmpp, that is the question. http://www.opensourcery.co.za/2009/04/19/to-amqp-or-to-xmpp-that-is-the-question/.

[128] C. Kalt. Internet Relay Chat: Architecture. RFC 2810 (Informational), April 2000.

[129] C. Kalt. Internet Relay Chat: Channel Management. RFC 2811 (Informational), April 2000.

[130] C. Kalt. Internet Relay Chat: Client Protocol. RFC 2812 (Informational), April 2000.

[131] C. Kalt. Internet Relay Chat: Server Protocol. RFC 2813 (Informational), April 2000.

[132] J. Karneges and P. Saint-Andre. Xep-0047: In-band bytestreams. XMPP Standard Proposal, June 2012.

[133] J. Kelsey, J. Callas, and A. Clemm. Signed Syslog Messages. RFC 5848 (Proposed Standard), May 2010.

[134] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001. Obsoleted by RFC 5321, updated by RFC 5336.

[135] J. Klensin. Simple Mail Transfer Protocol. RFC 5321 (Draft Standard), October 2008. Updated by RFC 7504.

[136] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151.

[137] D. Levi, P. Meyer, and B. Stewart. Simple Network Management Protocol (SNMP) Applications. RFC 3413 (INTERNET STANDARD), December 2002.

[138] C. Lonvick. The BSD Syslog Protocol. RFC 3164 (Informational), August 2001. Obsoleted by RFC 5424.

[139] Antonio Magnaghi, Takeo Hamada, and Tsuneo Katsuyama. A wavelet-based framework for proactive detection of network misconfigurations. In *Proceedings of the ACM SIGCOMM Workshop on Network Troubleshooting: Research, Theory and Operations Practice Meet Malfunctioning Reality*, NetT '04, pages 253–258, New York, NY, USA, 2004. ACM.

[140] K. McCloghrie. An Administrative Infrastructure for SNMPv2. RFC 1909 (Historic), February 1996.

[141] K. McCloghrie and J. Galvin. Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1447 (Historic), April 1993.

[142] K. McCloghrie and M.T. Rose. Management Information Base for network management of TCP/IP-based internets. RFC 1066, August 1988. Obsoleted by RFC 1156.

[143] K. McCloghrie and M.T. Rose. Structure and identification of management information for TCP/IP-based internets. RFC 1065 (INTERNET STANDARD), August 1988. Obsoleted by RFC 1155.

[144] F. Miao, Y. Ma, and J. Salowey. Transport Layer Security (TLS) Transport Mapping for Syslog. RFC 5425 (Proposed Standard), March 2009.

[145] P. Millard, P. Saint-Andre, and R. Meijer. Xep-0060: Publish-subscribe. XMPP Standard Proposal, July 2010.

[146] M. Miller. Xep-0050: Ad-hoc commands. XMPP Standard Proposal, October 2015.

[147] Tony Mobily. Nagios vs. icinga: the real story of one of the most heated forks in free software. http://www.freesoftwaremagazine.com/articles/nagios_and_icinga.

[148] Jack Moffitt. Thoughts on scalable xmpp bots. http://metajack.im/2008/08/04/thoughts-on-scalable-xmpp-bots/.

[149] T. Muldowney. Xep-0027: Current jabber openpgp usage. XMPP Standard Proposal, March 2014.

[150] Kenneth E Nawyn. A security analysis of system event logging with syslog. *SANS Institute, no. As part of the Information Security Reading Room*, 2003.

[151] D. New and M. Rose. Reliable Delivery for syslog. RFC 3195 (Proposed Standard), November 2001.

[152] J. Nishes. Interoperability in monitoring and reporting systems. 2012.

[153] Tobias Oetiker. Multi router traffic grapher homepage. http://oss.oetiker.ch/mrtg/.

[154] Tobias Oetiker. Round robin database tool homepage. http://oss.oetiker.ch/rrdtool/.

[155] Tobias Oetiker. Smokeping homepage. http://oss.oetiker.ch/smokeping/.

[156] Tobias Oetiker. Smokeping master/slave mode. https://oss.oetiker.ch/smokeping/doc/smokeping_master_slave.en.html.

[157] J. Oikarinen and D. Reed. Internet Relay Chat Protocol. RFC 1459 (Experimental), May 1993. Updated by RFCs 2810, 2811, 2812, 2813, 7194.

[158] A.J. Oliner, A. Aiken, and J. Stearley. Alert detection in system logs. In *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, pages 959–964, Dec 2008.

[159] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. Spins: Security protocols for sensor networks. *Wirel. Netw.*, 8(5):521–534, September 2002.

[160] J. Postel. Simple Mail Transfer Protocol. RFC 788, November 1981. Obsoleted by RFC 821.

[161] J. Postel. Simple Mail Transfer Protocol. RFC 821 (INTERNET STANDARD), August 1982. Obsoleted by RFC 2821.

[162] R. Presuhn. Management Information Base (MIB) for the Simple Network Management Protocol (SNMP). RFC 3418 (INTERNET STANDARD), December 2002.

[163] R. Presuhn. Transport Mappings for the Simple Network Management Protocol (SNMP). RFC 3417 (INTERNET STANDARD), December 2002. Updated by RFCs 4789, 5590.

[164] R. Presuhn. Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). RFC 3416 (INTERNET STANDARD), December 2002.

[165] Steven B Roosa and Stephen Schultze. The "certificate authority" trust model for ssl: a defective foundation for encrypted web traffic and a legal quagmire. *Intellectual Property & Technology Law Journal*, 22(11):3, 2010.

[166] P. Saint-Andre. Xep-0001: Xmpp extension protocols. XMPP Standard Proposal, July 2001.

[167] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), October 2004. Obsoleted by RFC 6120, updated by RFC 6122.

[168] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 3921 (Proposed Standard), October 2004. Obsoleted by RFC 6121.

[169] P. Saint-Andre. Xep-0128: Service discovery extensions. XMPP Standard Proposal, October 2004.

[170] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120 (Proposed Standard), March 2011. Updated by RFC 7590.

[171] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 6121 (Proposed Standard), March 2011.

[172] P. Saint-Andre. Xep-0045: Multi-user chat. XMPP Standard Proposal, February 2012.

[173] P. Saint-Andre. Xep-0114: Jabber component protocol. XMPP Standard Proposal, January 2012.

[174] P. Saint-Andre and T. Alkemade. Use of Transport Layer Security (TLS) in the Extensible Messaging and Presence Protocol (XMPP). RFC 7590 (Proposed Standard), June 2015.

[175] J. Salowey, T. Petch, R. Gerhards, and H. Feng. Datagram Transport Layer Security (DTLS) Transport Mapping for Syslog. RFC 6012 (Proposed Standard), October 2010.

[176] L.J.T. Stout. *Kestrel: Job Distribution and Scheduling Using XMPP*. BiblioBazaar, 2012.

[177] Z. Su. Distributed system for Internet name service. RFC 830, October 1982.

[178] Niklas Thörne. Xmppmote: Remote server administration via xmpp. https://github.com/nthorne/xmppmote.

[179] Gregory Trubetsky. Time series accuracy - graphite vs rrdtool. http://grisha.org/blog/2015/05/04/recording-time-series/.

[180] Michael Trunner. Syslogbot: forward syslog messages. https://github.com/trunneml/SyslogBot.

[181] United States. General Services Administration. *Telecommunications: Glossary of Telecommunication Terms*. General Services Administration, Washington, DC, USA, August 1996. Federal Standard 1037C.

[182] John R Vacca. *Public key infrastructure: building trusted applications and Web services*. CRC Press, 2004.

[183] Alex van den Bogaerdt. Rrd data point consolidation: Minimum, average, maximum. http://vandenbogaerdt.nl/rrdtool/min-avg-max.php.

[184] Ronald van der Pol and Freek Dijkstra. Data exchange between network monitoring tools. AIMS, 2009.

[185] P. Waher. Xep-0325: Internet of things - control. XMPP Standard Proposal, April 2014.

[186] P. Waher. Xep-0337: Event logging over xmpp. XMPP Standard Proposal, January 2014.

[187] P. Waher. Xep-0323: Internet of things - sensor data. XMPP Standard Proposal, March 2015.

[188] G. Waters. User-based Security Model for SNMPv2. RFC 1910 (Historic), February 1996.

[189] B. Wijnen, R. Presuhn, and K. McCloghrie. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). RFC 3415 (INTERNET STANDARD), December 2002.

[190] R. Zakon. Hobbes' Internet Timeline. RFC 2235 (Informational), November 1997.