A Platform for End-Host-based Active Measurements on the Internet

Dissertation zur Erlangung des Grades
eines Doktors der Naturwissenschaften (Dr. rer. nat.)
am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

von

Michael Faath, M.Sc.

Berlin
2016

Erstgutachter: Prof. Dr.-Ing. Jochen Schiller
Zweitgutachter: Prof. Dr. rer. nat. Rolf Winter

Tag der Disputation: 22.12.2016

Selbstständigkeitserklärung

Ich versichere hiermit, diese Arbeit selbstständig verfasst und nur die angegebenen Hilfsmittel und Hilfen verwendet zu haben. Ich versichere außerdem, dass diese Arbeit nicht in einem früheren Promotionsverfahren eingereicht wurde.

Ort, Datum Michael Faath

Abstract

The size and complexity of the Internet grow continuously. Billions of interconnected devices communicate over this large network of networks, transporting a steadily increasing amount of traffic. As the Internet grows, the importance of it as a vital telecommunications infrastructure is increasing as well. Given its role and importance, it is somewhat worrisome that an accurate status of the operational state of the Internet cannot reasonably be obtained. Many interesting questions about the Internet and its performance are difficult to answer and it would require a large number of vantage points to answer some of them. There is no complete "map" of the Internet and when problems occur, it is often difficult to identify their source, in particular in an automated fashion.

Measurement platforms could continuously monitor vital functions of the Internet and help to detect, locate and solve problems in the network. There is a great interest from Internet Service Providers (ISPs), end-users, and content providers to be alerted quickly if something is not working as expected, to pinpoint problems, and even to fix a problem before it manifests in some kind of performance problem or outage. Therefore, it would be beneficial to have a standardized way to execute distributed measurements on the Internet.

Multiple organizations and projects, for example the Large-Scale Measurement of Broadband Performance (LMAP) working group of the Internet Engineering Task Force (IETF) or the EU project mPlane, worked and still work on the standardization of measurement platforms. Those efforts make it possible to coordinate and execute comparable measurements on distributed probes, using common interfaces that allow different vendors of probes and other components of such measurement platforms to interoperate. GLIMPSE (Global Internet Monitoring and Probing System) is a concrete example of a platform for active measurement campaigns, which was developed alongside currently evolving standards. This platform can schedule measurement campaigns on a large number of probes deployed on common devices of end-users. The data produced by these measurements can be used to find and analyze anomalies in the Internet.

This dissertation gives an overview of standardization efforts, tools, and projects in the field of platforms for active measurements on the Internet. GLIMPSE, a platform developed as part of this dissertation, is used to describe the characteristics, design, implementation and limitations of such platforms. A *glimpse* into the possible

research and the obtainable results is given by looking at real measurement campaigns realized with this system. One such campaign shows for example how a popular method widely used by end-users for measuring network access rates at home is flawed.

GLIMPSE was build at a time of active standardization in the field of large-scale measurement platforms. To give researchers and developers data and advice as a basis to decide if it is worth to use still evolving pre-standards in their work, an analysis of the document process within the IETF was performed. Also, the main challenges when designing and implementing a platform for measurements with a large number of end-users have been identified and analyzed.

GLIMPSE probes rely solely on active measurements. In order to showcase the research potential of passive measurements on end-device probes, an analysis of broadcast and multicast traffic captured passively on such devices has been carried out. The results of this research was brought to the attention of the Internet Area working group of the IETF and has been adopted by that group as a working group item at the time of this writing. The respective Internet draft lists considerations for broadcast and multicast protocol designers.

Acknowledgments

My first thank you goes to Prof. Dr. Jochen Schiller, head of the Department of Mathematics and Computer Science of the Freie Universität Berlin, and supervisor of this dissertation. Thank you for the opportunity to write my dissertation at your department while working in a research project at the University of Applied Sciences Augsburg, for the feedback meetings and the trust to let me work freely in my field of research.

As head of the Networking Group at the University of Applied Sciences Augsburg Prof. Dr. Rolf Winter supported this work and was my first address for questions and problems. Thank you for the countless meetings, the invaluable feedback and insights, the possibility to work with you, the great people at the University and in the mPlane research project, and for the many opportunities to attent conferences, workshops and meetings.

Finally, a big thank you to my family and friends who supported me throughout the time of my research. Special thanks to Jessica, Benjamin, Christoph, Fabian and Thomas for their helpful feedback, but also for their patience with me when work got stressful. Hagen, I would not be where I am today if it was not for you. Thank you for posting the offer for my doctoral position on your Facebook page and for introducing me to Prof. Dr. Winter. Also thank you to my fellow doctoral candidates, especially Martin, who was always available to talk about research, formalities and the typical misery only known by other doctoral candidates.

This work has received funding from the European Union under the FP7 Grant Agreement n. 318627, project “mPlane”.

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

ASN Autonomous System Number

BRAS Broadband Remote Access Server

BTC Bulk Transfer Capacity

DDoS Distributed Denial of Service

DNS Domain Name System

ECDF Empirical Cumulative Distribution Function

GLIMPSE Global Internet Monitoring and Probing System

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IETF Internet Engineering Task Force

IPPM IP Performance Metrics

IP Internet Protocol

ISP Internet Service Provider

LMAP Large-Scale Measurement of Broadband Performance

MONROE Measuring Mobile Broadband Networks in Europe

MP Measurement Peer

NTP Network Time Protocol

NAT Network Address Translation

NAS Network Attached Storage

RTT Round-Trip Time

SDK Software Development Kit

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TTL Time-to-Live

UDP User Datagram Protocol

UPnP Universal Plug and Play

URL Uniform Resource Locator

UTC Universal Time Coordonné

1 Introduction

The Internet is a massive and complex distributed system. Billions of devices participate as end-hosts, as content providers or act as intermediate nodes within this network. The number of smartphone users alone is expected to surpass two billion in 2016 [eMa]. This impressive scale alone illustrates the importance of the Internet, but also the way the Internet is used and by whom speaks for its significant role in society. Taking Germany as an example, according to an online survey carried out by ARD and ZDF (the two largest state-funded television broadcasters), 79.5% of the people over the age of 14 use the Internet, 63.1% on a daily basis [ARD]. The Federal Statistical Office of Germany determined that 88.2% of all households in Germany have access to the Internet [Bun].

These numbers show that the availability of the Internet itself and the services used is of great interest to the public, service providers, and regulator alike. Albeit not being officially regarded as a critical infrastructure such as the power grid or the public road system in most countries, the Internet clearly plays a vital role. It is therefore of great importance to know the current state of the Internet to detect problems, uncover affected areas, and determine the cause of these issues.

Motivation

When talking about network problems and root cause analysis, multiple aspects have to be taken into consideration. One main problem is to realize in the first place that something is not working properly and there is an actual issue in the network. Some kind of operational oversight is necessary to achieve this, as the Internet does not have a built-in monitoring component. After initially detecting an issue, it might be necessary to investigate further to classify and fully understand the problem and what caused it. To find out who is affected, but also who is accountable, the source of the malfunction needs to be determined. This source can for example be in the network of an Internet Service Provider (ISP), a network transit provider, a content provider or an end-user, each with different capabilities in detecting and

fixing problems. Once an issue is identified, it might be necessary to communicate details between these parties for further investigation. To automate this process, not only the means of communication and coordination need to be specified, but also the measurements to find and analyze problems.

These aspects, and many more not listed here, make it obvious that it is no easy task to detect and locate problems on the Internet and to apply (semi-)automatic reasoning to them. To comprehend and reproduce issues seen on the edge of the network, it is essential to run measurements near or directly on end-users devices. Users are often the first to experience problems, for example by witnessing an increase in the networks latencies or a reduction of the available bandwidth. Therefore, it would be beneficial for network operators to be able to measure from end-to-end: from the end-host of the user through the Internet to the end-host of the service provider. Also, as the Internet is a distributed system with an enormous number of participants, different vantage points from within and from the edge of the network are necessary to fully understand problems. One solution to execute measurements from multiple vantage points and to coordinate them from a few central points of control is the deployment of a distributed measurement platform. Such a platform allows to specify measurements and to execute them on one or many participating devices, so called measurement probes. These probes might be located directly on end-systems as an installed application, on home gateways, or on dedicated devices in the home network which are solely used for measurements. Results gathered could be sent back to a central component of the platform, or they could be used on the measuring device to drill down further by executing other measurements in case a problem was detected. The actual measurement tools used can be classified as active (they generate the measurement traffic), passive (they listen passively to traffic generated by others), or hybrid (a combination of both). This work mainly concerns itself with active measurements for multiple reasons. For one thing, deploying a passive measurement probe which listens to user generated data in the premises of the end-user is typically seen as intrusive and brings up serious privacy concerns. Also, when using a software-only solution, listening passively normally requires administrative privileges on the device. This is not possible on every device. For example, mobile phones do not allow to execute applications with administrative access without rooting the device. This would reduce the acceptance by users and consequently the number of participants considerably which limits the utility of the platform. For another thing, to capture the real experience and problems of

end-users, the point of measurement needs to be located at the same location as the user. In contrast to an end-host based approach, the point of measurement could also be on the path (in-band) between one or multiple systems. ISPs are already able to measure from within their own networks, but a (standardized) way to measure from the edge of the network is missing and needs to be researched.

Using active measurements also brings some disadvantages which need to be taken into consideration by a measurement platform. Obviously, generating measurement traffic increases the overall load in the network. This is especially important to remember in cases when there are already problems in the network present. By trying to detect an issue, the measurement could make the situation even worse. Another factor to consider is cross traffic. When executing measurements, another application on the same host or even on another device in the same network can impact the results of the measurement. An example where this is of importance is when trying to measure the overall bandwidth of the Internet connection. Many Internet contracts that are sold nowadays do not specify a fixed bandwidth value but give an "up to" and a minimum guaranteed bandwidth. Taking one German ISP as an example, up to 100,000 kbit/s download bandwidth is advertised for their product but only a minimum of 54,000 kbit/s is guaranteed and a "normal" bandwidth of 88,600 kbit/s is available for 80% of the customers [Tel]. End-users often try to measure their available bandwidth with tools like Flash-based browser plug-ins which do not take cross traffic or other factors like the signal strength of the WiFi or the system load into consideration. A measurement platform with probes in the network of the end-user could execute such measurements and provide the customer as well as the ISP a way to determine the actual bandwidth and many other useful metrics.

This dissertation makes multiple contributions. First of all, it provides an overview and a classification of already existing work in the area of active measurement platforms for the Internet. Based on this, an analysis of the common components of such platforms is presented and put into the context of current standardization efforts. Especially for measurements from and in the homes of end-users many factors need to be taken into consideration. With the help of a measurement platform developed for this dissertation, the difficulties encountered while specifying and implementing these kinds of platforms are shown. To demonstrate the research possible, results from measurement experiments conducted with the developed platform are analyzed. The primary focus of this was bandwidth estimations from the end-user perspective,

but also the server selection process of a popular bandwidth estimation service and its resulting bandwidth estimation accuracy. The results indicate that today's widely used methods for measuring access speeds in end-users homes are flawed. Often, the underlying technologies (Flash plugins in a web-browser) do not produce accurate results and lead to wrongly estimated download- and upload rates. Another major part of this document is concerned with the problems of research in a field with active but unfinished standardization efforts. Amongst other things, it is shown that pre-standards often change too rapidly and too unpredictably to be a useful basis for research projects with a short or medium lifetime. For this, the IETF document life cycle is thoroughly analyzed. Finally, as part of an experiment involving passive measurements, the question of anonymity of participants in a large broadcast domain such as campus networks is discussed. The results of an experiment carried out over a period of six months which combined information from multiple broadcast and multicast protocols showed that, given today's protocol landscape, it is possible to construct the social graph of the networks user base without special privileges. This experiment demonstrated that there is a need for guidance, principles and rules for broadcast/multicast protocol design. This research lead to an ongoing standardization effort in the IETF to make sure broadcast protocol designers take special care while designing and implementing such protocols.

Context of this work

Multiple research projects, companies, institutions, and even standards organizations are working on this topic. In the past few years, two working groups of the IETF have been formed which are directly involved with network measurements and network measurement platforms: the IP Performance Metrics (IPPM) and the Large-Scale Measurement of Broadband Performance (LMAP) working groups. IPPM works on "standard metrics that can be applied to the quality, performance, and reliability of Internet data delivery services and applications running over transport layer protocols" [IETa], while LMAP focuses on standards for performance measurement platforms by providing an information model and data models for the participating components [IETb].

In addition to these standardization efforts, there are already some deployed measurement platforms with thousands of users out there. Those range from research-driven measurement projects to generalized commercial solutions to ISP-specific products. One of the most prominent examples for a research platform is the RIPE

Atlas project [RIP]. It allows its over 25,000 users to schedule active measurements on the almost 10,000 deployed probes. The project uses dedicated single board computers to execute simple measurements like ping and traceroute. Users earn credits by deploying a probe in their own network, these credits can be spent to schedule measurements on the platform.

A popular commercial solution for bandwidth estimation is Speedtest.net, a service developed by Ookla. It provides on-demand measurements for end-users in contrast to schedule-based measurement platforms. The Flash-based website allows users to measure the latency and upload/download speed. Many ISPs cooperate with Ookla and provide a measurement end-point in their own network to improve the results for their customers. There is existing research (for example in [CBS12] or [SB12]) looking at this approach of measuring bandwidth which is summarized and continued in this document. The focus here was to evaluate the accuracy of the results of Speedtest.net [Ookb] by doing the same measurements the Flash solution does using GLIMPSE probes and comparing the results.

Another project concerning itself with the specification, implementation and deployment of a measurement platform is mPlane. The work on this thesis started as part of this project.

mPlane and GLIMPSE

The mPlane research project—funded within the 7$^{\text{th}}$ Framework Programme for Research and Technological Development (FP7) of the European Commission—tried to specify and implement "a Distributed Measurement Infrastructure to perform active, passive and hybrid measurements". The mPlane consortium consisted of 16 partners including universities, research institutes, ISPs and hardware vendors. One of those partners, the University of Applied Sciences Augsburg (HSA), focused on the development of an active end-user-based measurement platform named GLIMPSE (Global Internet Monitoring and Probing System), which is briefly described in the next paragraphs. Chapter 5 describes the design and development of this platform as well as the research done with it.

GLIMPSE is an end-host-based network measurement platform for active measurements. The probe is written in C++ with the help of Qt which makes it cross-platform by design. The architecture was based on draft working group documents (meaning they are not already a standards document) of LMAP at first and later on changed to the mPlane protocol specification with some additions from LMAP.

A detailed description of the design and implementation can be found in chapter 5, the following paragraphs provide a brief summary of the platform as a whole.

The original intend for GLIMPSE was to allow users to execute measurements when they encounter problems while using the Internet. This focus has shifted during development to make sure every component is tested before end-users can use GLIMPSE. Therefore, a more centralized system was deployed where measurements for all probes or a specific group of probes can be scheduled from a single point. Gradually, more possibilities for end-users to use the system directly were introduced, beginning at the end of 2015 with an interface to schedule latency measurements for the owned probes.

The decision of making the probe cross-platform was made to allow as many users as possible to use GLIMPSE and have the probe run on as many different devices (and architectures) as possible. Another important design goal was to run GLIMPSE without root privileges on end-systems. This restriction is necessary because mobile platforms like Android do not allow the execution of software as root on a normal (meaning unrooted) device. Unfortunately, this limits the power of the tool. Most notably, this makes it impossible to execute passive measurements, which classifies GLIMPSE as an active measurement platform. A passive measurement probe on devices of end-users would also bring up some serious privacy concerns. It would be more difficult to find users willing to install a crowed-sourced measurement platform if the probe reads traffic generated by the user. Other examples of limitations caused by missing privileges are that it is not possible to send Internet Control Message Protocol (ICMP) packets as an unprivileged user on every Linux system or to read some information from the device like WiFi statistics on iOS.

GLIMPSE, being a software-only approach, brings some other disadvantages. Normally, the probe does not run exclusively on a device, there is always the possibility that measurements were influenced by other software running on the same machine. Some means to limit such possibilities or at least be able to detect if a measurement got tainted were applied.

An overview of the measurement tools included in GLIMPSE is given here:

- Latency estimation using either
  (a) Transmission Control Protocol (TCP) packets,
  (b) User Datagram Protocol (UDP) packets,
  (c) the ping-tool which is installed on the system which often can use ICMP

packets without root privileges,

- Domain Name System (DNS) and reverse DNS lookups,
- bandwidth estimation techniques like Hypertext Transfer Protocol (HTTP) download
- UPnP lookup to collect information about devices in the local network

The GLIMPSE measurement platform is a unique combination of (a) the ability to run centrally specified measurements on (b) multiple, heterogeneous probes designed to be deployed on end-user devices using (c) a well-specified protocol and interfaces with (d) an externalized logic to schedule new measurements based on previous results. Given this, GLIMSPE enables a number of novel research opportunities.

Document structure

This work provides a description and comparison of already existing platforms, the current state of the standardization process, and an overview of already published research in the field of network measurements in chapter 2. A demonstration of the problems faced when using Internet standards-to-be in research work is presented in chapter 3. It contains a study of the standardization process of the IETF based on an analysis of the IETF's datatracker database. A discussion of the challenges faced when trying to incentivize end-users to participate in a measurement platform or specific campaign is given in chapter 4. In chapter 5 the developed GLIMPSE measurement platform is described, while chapter 6 presents and discusses the results of conducted network experiments with this platform. In contrast to the active measurements which are described and used in most parts of this work, chapter 7 covers the results of an experiment with passively obtained data, showcasing the benefit of potential extensions to the probes. Finally, chapter 8 concludes this document by summarizing the main findings and giving an outlook on future research opportunities.

The chapters 3, 4, and 7 have already been published in shorter forms in peer reviewed conferences and a workshop [FWW15] [FW15] [FWW16].

2 Related Work

The majority of this work was done in the context of the mPlane project [mPlc]. Therefore, most terminology used here was specified and documented in the project's public deliverables [mPla]. For this reason, the mPlane architecture and related work is described first in this chapter. The components and terminology as defined by mPlane form the basis for the rest of this document. When describing work from other platforms or projects, the respective terms as defined by mPlane are used whenever possible.

2.1 mPlane

> The Internet is the first thing that humanity has built that humanity doesn't understand, the largest experiment in anarchy that we have ever had.
>
> Eric Schmidt – ex Google Exec. Chairman

mPlane was a research project active between 2012 and 2016 with the goal to build "an Intelligent Measurement Plane for Future Network and Application Management", as the complete title of the project states. It was funded by the European Commission in the context of the 7$^{\text{th}}$ Framework Programme for Research and Technological Development (FP7). The fundamental idea behind the project was to "design and demonstrate [...] A distributed infrastructure for network measurements, [...] which perform passive and active measurements, continuously or on-demand, at a wide variety of scales [...] with built-in support for iterative measurement and automated iteration"[Mar].

2.1.1 Consortium

The mPlane consortium consisted of 16 partners with Prof. Dr. Marco Mellia from Politecnico di Torino as project coordinator. The list of partners included three ISPs, six research centers, five universities and two small enterprises. A complete list of all partners can be found in table 2.1. Note that the company Távközlési Zrt. left the project within the first year and was replaced by the Italian ISP FASTWEB.

	Organization name	Short name	Country
1	Politecnico di Torino	POLITO	IT
2	Fondazione Ugo Bordoni	FUB	IT
3	SSB Progetti	SSB	IT
4	Telecom Italia	TI	IT
5	Alcatel-Lucent Bell Labs	ALBLF	FR
6	EURECOM	EURECOM	FR
7	Telecom Paritech	ENST	FR
8	NEC Europe LTD	NEC	UK
9	Telefonica Investigacion Y Desarrollo Sa	TID	ES
~~10~~	Invitel Távközlési Zrt.	INV	HU
11	Netvisor	NETVISOR	HU
12	Forschungszentrum Telekommunikation Wien	FTW	AT
13	Fachhochschule Augsburg	FHA	DE
14	Universite de Liege	ULG	BE
15	Eidgenoessische Technische Hochschule Zurich	ETH	CH
16	Alcatel-Lucent Bell Nv	A-LBELL	BE
17	FASTWEB	FW	IT

Table 2.1: The mPlane consortium

While many pieces of the project work were a collaborative effort, such as the definition of the mPlane protocol and the architecture, the work on GLIMPSE was mainly done at the University of Applied Sciences Augsburg (partner 13) and forms the core of this dissertation.

2.1.2 Architecture and Protocol

GLIMPSE as an instantiation of the mPlane architecture implements all its architectural components. An overview of these entities and their interactions can be found in figure 2.1.

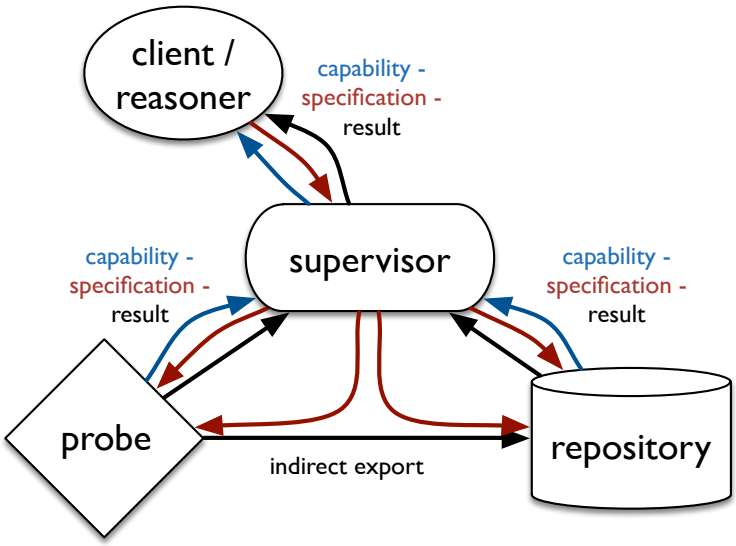


Figure 2.1: Overview of the mPlane architecture and its entities [TMF+15b, page 7]

The lower left part of 2.1 shows a **probe**. This is the entity which generates measurement data by executing measurements now or in the future. The **supervisor** in the figure is the entity which (among other things) controls one or multiple probes. There are three basic types[1] of messages exchanged between all entities [TMF+15b, page 20ff.]:

(a) A **capability** details an operation a component can perform. A probe can expose multiple capabilities to signal a supervisor which kind of measurements can be executed, which parameters can be set and which results to expect from such a measurement.

(b) A **specification** is the actual imperative to execute a capability. Basically, it is just a capability with filled-in parameters. A supervisor might sent one or multiple specifications to a previously offered capability by a probe.

[1] There are in fact some other types which are ignored here for the sake of simplicity

(c) A **result** is—as the name implies—the outcome of a measurement. A probe normally sends a result to the supervisor after successfully executing a specification.

An example of a capability can be found in listing 2.1, which shows a real capability advertised by the GLIMPSE probe described in a later chapter.

The first two lines identify this capability as a measurement operation with the name "glimpse-dnslookup". With the help of the label, a specification can reference a received capability. The list of parameters shows the available options for this measurement. In this example, a URL for which this measurement should return the DNS records and the DNS resolver to use. It is possible to constrain the values for the parameters, for example by giving one, multiple or a range of allowed values. In this case, the asterisk specifies that every value is allowed. The list of results determines which result values the measurement can return. A specification can take this capability, fill in the parameters as necessary and set the result fields which should be returned. Some parameters are optional as they can have default values, such as the system-wide local DNS resolver.

Before taking a look at the other entities of the architecture, figure 2.2 depicts the general relationship between two parties in the architecture.

Entities can be differentiated in **components** which make data available and **clients** which request data from components. Typical components are probes which execute measurements and **repositories** which store and provide measurement results. Components define all available operations by their capabilities while clients invoke those by sending specifications. From an architectural point of view, probes and repositories do not differ all that much when it comes to measurements: probes provide capabilities and receive specifications to invoke these capabilities at given future points in time while repositories do the same for past measurements (meaning already measured and stored results) [Bri, Slide 3]. Repositories additionally provide capabilities to store measurement results, either from a supervisor or directly from a probe, by something which the architecture calls "indirect export" as seen in figure 2.1. This functionality specifies how to send data directly to the repository with a non-mPlane mechanism to take load off the supervisor, mainly used in scenarios with a large number of probes.

The remaining entities shown in figure 2.1 are the client and reasoner. A client (not to be confused with the more general architectural notion of a client in 2.2) is a piece of software which allows a human to see the capabilities, define specifications and

```
"capability": "measure",
"label": "glimpse-dnslookup",
"parameters": {
    "destination.url": "*",
    "glimpse.dns.server": "*"
},
"registry": "https://www.measure-it.net/static/
   glimpse_registry.json",
"results": [
    "glimpse.dns.record-name",
    "ttl",
    "glimpse.dns.record-value",
    "glimpse.dns.record-type"
],
"version": 0
```

Listing 2.1: An example of a mPlane capability

look at results. The supervisor runs as an intermediate by delegating the received capabilities and results from probes to the client (in which case the supervisor actually is a component when viewed from the client side and a client when viewed from the probe side) and the other way around for the specifications. Instead of a client, there can also be a **reasoner** which is an unsupervised component that implements a (semi-)automatic process of scheduling measurements using some kind of learning technique. A reasoner, in essence, is an intelligent agent to make use of the collection of probes and their capabilities to learn about the network and to identify certain properties (e.g. to identify the root cause of a network anomaly). The original plan of mPlane was to provide a single reasoner which works as the

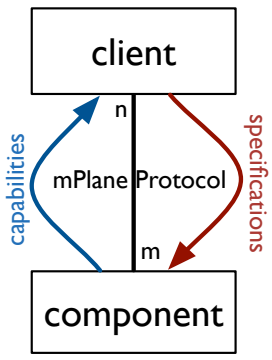


Figure 2.2: Simplest relationship in the mPlane architecture [TMF+15b, page 8]

central intelligence of the system. While working on the architecture and on a larger collection of use cases, the consortium realized that one such reasoner is to complex to specify and build within the project's lifespan. It was decided to focus on the architecture and provide a capable Software Development Kit (SDK) with which exemplary software demonstrators were built to give other researchers and developers the tools and guidance necessary to build reasoners for specific use cases.

One key architectural principle of mPlane is that everything a component can do is completely described by its capabilities [Bri, Slide 5] which state the operation, parameters and result fields (schema-centric measurement definition). Another architectural principle—weak imperativeness—states that capabilities are not guaranteed but are only provided on the basis of best-effort [TMF+15b, page 9f]. This also means that every message sent in the mPlane protocol always contains all information necessary to make it comprehensible in itself. This implies, for example, that every result also contains the complete specification with all parameters used to generate it.

An example flow of messages within an mPlane-compliant system can be seen in figure 2.3. A component advertises its invokable operations via capabilities to a

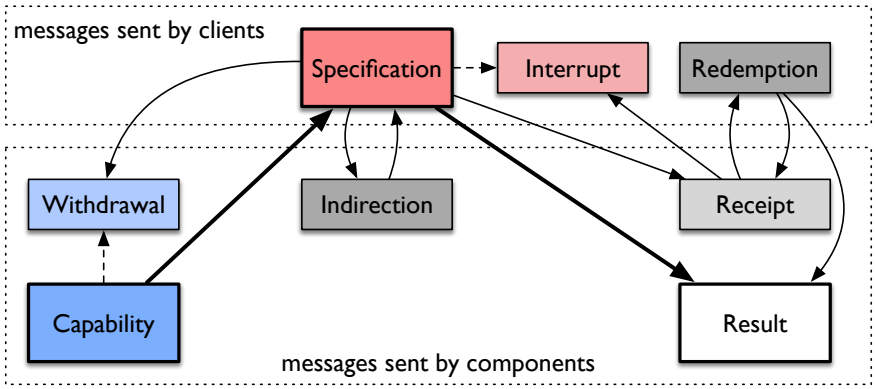


Figure 2.3: Potential message flow [TMF+15b, page 14]

client which it can use to generate one or more specifications. A specification can be sent back to the component to be executed there which will ultimately generate a result message, which again is sent back to the corresponding client. This picture also shows some additional message types used in the protocol:

(a) A withdrawal can be sent by a component to tell clients that a capability is no longer valid.

(b) Interrupts are used by clients to signal a component to abort an operation previously initiated with a specification.

(c) If a client wants to retrieve a result from a component which has not finished the operation yet, the component can answer with a receipt, telling the client to ask again later by sending a redemption message with a reference to the receipt.

(d) An indirection can be used by a component to delegate the responsibility for a specification to another component.

An example for a specification which was generated based on the previously shown capability is shown in listing 2.2. The main differences to the specification are a filled-in parameter (line 4), a unique token (line 6) to identify the corresponding results for this measurement later, and a timing information to specify when to execute this measurement (line 7)—in this case as soon as possible. mPlane calls the value of the "when" field the **temporal scope**. In case of a specification for a measurement, it defines when to execute it. When querying a repository, it states a time in the past for which to retrieve results. The temporal scope is fairly flexible as it can be an absolute point in time or relative to one. It may also have an optional period specifying that a measurement should be repeated periodically. Some examples for the temporal scope are shown below:

now Only once as soon as possible

now + 10m / 30s As soon as possible, repeat every 30s for the next 10 minutes.

2009-04-04 04:00:00 Only once at the given time-stamp

repeat now ... future / 1h now + 5m / 1s This scope has an inner and outer temporal scope. The inner scope determines to execute every second for five minutes, while the outer scope states when to run the inner scope. For a measurement this example would mean to run the measurement every second for five minutes and repeat this once an hour indefinitely.

when: repeat now ... future cron 0 0 * 1,2,3,4,5,6,7 1 * now + 5m This example runs every hour for five minutes on the first Monday of each month. The syntax is similar to the cron service typically found in Linux systems. The numbers identify the seconds, minutes, hours, days of the month, days of the

```
"specification": "measure",
"label": "glimpse-dnslookup-1",
"parameters": {
    "destination.url": "measure-it.net",
},
"token": "984f1b2050e5407abda6cd3e999b1a5d",
"when": "now",
"registry": "https://www.measure-it.net/static/
   glimpse_registry.json",
"results": [
    "glimpse.dns.record-name",
    "ttl",
    "glimpse.dns.record-value",
    "glimpse.dns.record-type"
],
"version": 0
```

Listing 2.2: An example of a mPlane specification

week and months in which to run this schedule (an asterisk determines to run it in every second/minute/etc.).

The last two examples show that the temporal scope can get quite complex and needs to be specified and tested carefully. Other examples as well as the complete syntax for the temporal scope can be found in [TMF+15b, Page 24ff].

Every mPlane message contains a registry which specifies the elements used in the message. For each element, the name, data-type (named "prim") and a description has to be given. By using the same registry, elements are comparable even if different measurements are being used, as the meaning is specified. It is possible to write a personal registry and inherit the elements of one or multiple other registries, which multiple partners in the project have done for their probes in cases where specific elements were needed.

2.1.3 SDK and Software

The mPlane SDK offers a toolkit to build mPlane-compliant clients and components. It provides methods to generate capabilities, specifications and results and has a reference implementation of a supervisor as well as working probes as examples. It is written in Python 3 and is available on GitHub as open-source software[2]. Multiple

[2] https://github.com/fp7mplane/protocol-ri

tools developed within the mPlane project make use of this SDK and are publicly available for further reference. Examples of these tools are Tstat[3] and Blockmon[4] as passive probes and Tracebox[5] and GLIMPSE[6] as active probes.

2.1.4 Further Project References

The core documents describing the mPlane project itself are the public deliverables. Most notable are the "mPlane Architecture Specification" [TMF+15b] and the "Final Implementation of Software and Libraries - Final Release" [TMF+15a] with a description of the SDK and a complete list of developed tools.

A large number of other publications with relation to mPlane were published by the partners during the project's life-time. Overall, 126 scientific publications were released. 18 of these were published in journals, 105 at conferences, and three book chapters were authored [Moh15]. Seven patents were filed and three RFCs published. The complete list of all publications is available on the mPlane public website[7].

2.2 Standardization

Given that the goal of a distributed measurement platform is to incorporate a large amount of possibly heterogeneous measurement end-points, repositories and potentially multiple supervisors, a standard data model and protocol are necessary to guarantee interoperability between the various components of the platform. Therefore, standardization efforts in this space are briefly described in the following sections.

2.2.1 Internet Engineering Task Force (IETF)

There are two working groups in the IETF relevant to measurements and measurement platforms.

Firstly, the Large-Scale Measurement of Broadband Performance (LMAP) working group which was started in 2013. The charter of the group states: "The [...] working group standardizes the LMAP measurement system for performance measurements

[3] https://www.ict-mplane.eu/public/tstat
[4] https://www.ict-mplane.eu/public/blockmon-node
[5] https://www.ict-mplane.eu/public/tracebox
[6] https://www.ict-mplane.eu/public/glimpse
[7] https://www.ict-mplane.eu/public/publications

of broadband access devices such as home and enterprise edge routers, personal computers, mobile devices, set top box, whether wired or wireless". Their goal is to make measurements, result-collection and the actual results from different measurement platforms compatible to each other (by using the LMAP standards). They developed a framework (RFC 7594 [EMB+15]), an information model [BEBS16] and a YANG-based data model [SB16]. Due to the nature of the process of developing an Internet standard only some aspects of the LMAP standards could be taken into consideration for the development of GLIMPSE. More details on this matter can be found in chapter 3 where the standardization process of the IETF is analyzed and chapter 5 which describes GLIMPSE in detail.

Secondly, the IP Performance Metrics (IPPM) working group works, amongst other things, on a registry for standard metrics for the Internet and measurements for these metrics. Similar to the LMAP working group, the results here were not used directly in GLIMPSE as the development is still ongoing. Also, for the case of standard metrics, the mPlane project has specified a small registry with interoperable metrics to be used within the mPlane project [mPlb], and a GLIMPSE registry which extends the mPlane registry also exists.

2.2.2 Broadband Forum

The Broadband Forum is an organization that has standardized very few protocols itself but relies on other organizations such as the IETF for this. It rather works on architectures, best practices, strategies, certification and marketing and leverages existing standards or attempts to influence standards defining organizations via liaison statements. The Broadband Forum is, contrary to the IETF, a membership-based organization, and work-in-progress documents are not open to the public. This makes it difficult to know what is being worked on as a non-member.

In the area of measurement platforms, the Broadband Forum publicly published their work in 2015 as Technical Report 304 [Bro15] which more closely defines service provider requirements and specifies requirements on components of the platform. That work, however, was performed in close collaboration with the IETF and is technically consistent with the work of the LMAP and IPPM working groups.

2.3 Other Research and Industrial Projects

There are numerous other projects which are related to the work presented here. Table 2.2 gives an overview of general purpose measurement platforms that use active measurements, but also some tools using a specific technology or focusing on a specific research topic. Only platforms and tools relevant for this work which have been active in the last three to five years have been included.

GLIMPSE is given as a reference at the top of this table and is described in chapter 5 in detail.

Dasu is a measurement platform developed and maintained by AquaLab (Northwestern University) [SOB+13]. As can be seen from the table, it is the most similar to GLIMPSE as it was developed to be run on end-users' devices. The first version was released as a plugin for a BitTorrent client, but the project has evolved and now has a standalone version running on desktop systems. The supported measurement methods include ping, traceroute, DNS lookup, HTTP GET and M-Lab's network diagnostic tool.

Project BISmark, a cooperation between Georgia Tech, Princeton University and M-Lab, is also a platform for measurements originating from the home of end-users [SBF14].

In contrast to GLIMPSE and Dasu, it was initially intended to be run on home gateways by providing an OpenWrt [Ope] firmware image. Today, the probe is also available for Linux and for older versions of Android (up to version 4.x). Amongst its provided set of measurements are multiple latency, traceroute and bandwidth estimation measurements.

SamKnows has the goal of "Providing accurate broadband performance data for consumers, governments and ISPs" [Sam]. It provides a number of so called Whiteboxes (hardware devices) for fixed-line and mobile broadband testing from the homes of end-users. Apps for versions of Android and iOS are available for some countries. SamKnows provides a great deal of whitepapers to describe the measurement platform and metrics used for testing [Sam15]. These metrics include multiple bandwidth-, latency- and packet loss-measurements as well as some methods related to specific services like Netflix, YouTube or the BBC iPlayer.

RIPE Atlas by RIPE NCC [RIP] and Archipelago (Ark) by caida [cHK+09] both rely only on dedicated hardware probes, in contrast to the other platforms. This allows them to be sure that running measurements are not tainted by unknown

	Name	Hardware probe	Software probe			Sourcecode
			Desktop	Mobile	Home gateway	
Platforms						
	GLIMPSE	no	Linux, Mac, Windows	Android, iOS	no	yes
	Dasu	no	Mac, Windows	no	no	on request
	Project BISmark	no	Linux	Android 4.x	OpenWrt	yes
	SamKnows	yes	no	Android, iOS	no	yes
	RIPE Atlas	yes	no	no	no	yes
	Archipelago	yes	no	no	no	no
Tools						
	Netradar	no	no	Android, iOS, ...	no	no
	Scamper	no	Linux, Mac, Windows	no	no	yes
	Glasnost	no	Website	no	no	yes
	ICSI Netalyzer	no	Website, Java	Android	no	no

Table 2.2: Measurement platforms and tools using active measurements

processes running on the same devices, which is also true for the Whiteboxes from SamKnows. Both platforms support latency and traceroute measurements. RIPE Atlas also includes SSL, DNS, NTP and HTTP measurements while Archipelago provides an interface for others to write measurements and include them into the platform. This was for example done with scamper [Luc]. Similar to GLIMPSE, these two platforms offer users the ability to perform measurements on their own. Ark provides a web-based interface to perform a selected subset of one-off measurements while Atlas provides an API to perform measurements. In addition, Atlas works on a credit based system. Users of the platform earn credits over time while the hardware probe they host is online, which they can in turn use to perform measurements of their own as a form of currency.

The tools section of the table lists tools that are not complete measurement platforms. They do not have a central measurement controller to schedule and distribute measurement instructions to multiple deployed instances of the tool. Nevertheless they do provide interesting capabilities related to the work presented here. The previously mentioned scamper, for example, is included into Archipelago as a measurement method. It can run by itself as a command line tool but can also be remotely executed on many devices with the help of the Ark platform. Netradar, written by the Aalto University, is an example of a mobile measurement app [Son13]. It is available for all common mobile platforms and faces the same problems GLIMPSE faced (as described later in detail). Methods here include speedtests, latency measurements, and the collection of statistics available on mobile phones like signal strength, base stations or operator information. Finally, Glasnost from the Max Planck Institute for Software Systems [Dis10] and ICSI Netalyzer from Berkeley University of California [KWNP10] are examples for measurements possible from within a web browser (both by utilizing Java). Both try to analyze the Internet connection for possible blockings from the ISP. Glasnost also tries to detect traffic shaping policies, while Netalyzr provides some DNS and latency measurements and a large number of other measurements.

3 A Cautious Look at Using Internet Standards-to-be in Research Work

The text written in this chapter was published as a paper at the IEEE Conference on Standards for Communications and Networking (CSCN) 2015 [FWW15]. This work summarizes the experience from working with the IETF LMAP working group. At the beginning of the development of GLIMPSE, it was intended to use the mPlane architecture while still being compatible as much as possible to the standards developed by LMAP. Adhering to an emerging standard was seen as a way to increase the chances of wider adoption later on, as GLIMPSE could serve as a standard component of an LMAP-based measurement system. However, it soon became apparent that the process of the IETF as a standards organization can be somewhat slow compared to the rapid changes needed while developing a prototype in a time-constrained project like mPlane. Also, depending on various and unforeseeable factors, the direction of a working group in the IETF can change drastically, for example by using (or changing to) a specific protocol or leaving certain aspects out of scope. Additionally, some components needed by a working group might be under change control of yet another working group, making the standardization process even more complicated. Therefore, it was not possible for GLIMPSE to fully implement LMAP. Only certain aspects could be used, as described later on. The pressing question this work tried to address was whether a continued involvement with the working group was likely to be fruitful for the work on GLIMPSE and it led to answering a much larger question: What time does it take for a standard to pass all stages of the standardization process when interacting with the IETF for research work, and are there any metrics with which progress can be estimated?

3.1 Abstract

Standardization of Internet protocols is usually a somewhat slow process. The reasons for this are manifold. Besides working out the protocol details, things like opposing stakeholder interests can prolong the consensus building process, new requirements might be introduced that require technical changes to the protocol, coordination across standards developing organizations (SDOs) might add delays just to name a few.

For potential users of a standard-to-be, the time to specify and implement it often stalls progress on projects which could have been finished far earlier using proprietary—but ultimately non-interoperable—implementations. For research work, interoperability is however not always an important concern. The overhead and delay of an SDO in these cases is typically a hard to calculate risk for a research project. It represents an external dependency for the work but there is only a finite amount of funding and time to finish the project. On the other hand, using standardized technology increases the likelihood that the output of the project is being used by external parties and the implementation experience can be valuable input to the standardization process.

In this chapter, we analyze the life cycle of recent Internet standards to provide researchers an insight into the Internet Engineering Task Force (IETF) standardization process duration. We evaluate different areas, document phases, working groups and other aspects of the standardization process. This allows researchers to better judge whether they want to employ standards-to-be in research work or engage with the IETF to specify protocols based on research prototypes.

3.2 Introduction

The trigger for the analysis described here was implementation experience of an Internet standard-to-be as part of a research project. In said project, a protocol was being developed when an Internet Engineering Task Force (IETF) working group was formed that was chartered to work on a very similar problem. The initial charter of that working group excluded some aspects that the research project felt important, but these exclusions seemed not prohibitive enough to not at least attempt to align with the IETF work where possible. The general idea was to keep developing the research protocol and data model in a way that, when the IETF had finished its

own protocol, the changes necessary to support the IETF protocol in the research prototype would only require small(ish) modifications to the already existing code.

In practice, a number of problems made the above difficult and ultimately we failed to do so. The IETF working group started out with developing an information model—an abstract representation of the information that a protocol needed to transport. The research project however was already implementing the protocol because it needed it for its experiments. That meant that the project already chose a protocol and encoding of the information model—a data model—while the IETF was one step behind. However, there was already one proposal for a protocol and data model and it was exactly what the project went for—a combination that seemed to be the obvious choice.

The information model developed slowly. At the time of this writing it is still not finished—two years after its first version. During this time, it underwent seven revisions. That is close to three and a half months between revisions, which often came with a number of changes and additions that were increasingly difficult to retrofit into the existing software that evolved at a faster pace. These revisions were typically just before an IETF meeting, so that document progress was basically clocked by the IETF three-meetings-per-year cycle.

Then, after a long period of time where no alternative protocol and data model were suggested, one was suggested and became the working groups choice relatively quickly. At this point, all effort of aligning the project's protocol was given up.

None of the above should be understood as criticism of the IETF process. All decisions made by the group had good and understandable reasons. Many documents in the IETF are updated solely before the meetings, because that is the point when the changes are being discussed in person. But this experience triggered a couple of questions. E.g. how long does it take on average for a document to become an Internet standard? How long does it take for a document to be adopted by a working group? Are there areas in the IETF that potentially work faster than others? These, and a couple of other questions will be answered in this chapter based on an analysis of the IETF datatracker database. The answers to these questions and the document history statistics hopefully give researchers better insights into what to expect when working with the IETF in terms of time commitment and document progress.

3.3 A Short Overview of the IETF Document Process

The IETF publishes standards as part of the Request for Comment (RFC) document series. RFCs are not all protocol specifications. They are categorized into e.g. informational, best current practice, experimental, Internet standard etc. to specify the nature of their content. Typically, these documents are the product of working groups—groups of people with expertise in a certain area working on a tightly scoped problem or protocol. Working groups in turn belong to an area, which is a collection of working groups belonging the same general field such as routing or transport.

Documents are submitted as Internet-drafts for review and typically undergo several revisions before being adopted by a working group of the IETF as a work item. Before a document is adopted, it is called an individual submission and the working group has not yet made a decision about whether the group will actively contribute to the development of it. Once a document has received enough support, it becomes a working group document and then the intent is to develop it further and ultimately to publish it as an RFC. These two distinct phases in the life cycle of a document are interesting as the first one is a matter of convincing the community of the necessity or usefulness of the document and the latter phase is then a matter of working out the details of the specification.

This description of the process is a simplified version of the reality of the IETF document process but sufficient for the purpose of this chapter. More details can be found in RFC 2026 [Bra96].

3.4 An Analysis of IETF Document Progress

We used the IETF datatracker database to perform our analysis. The IETF datatracker not only keeps track of documents but virtually any aspect of IETF process, activity and state including things such as meetings and working groups. The data can be accessed at `https://datatracker.ietf.org/` or since December 2014 via a machine-readable API. We experimented with the API at first but then decided to copy the complete database locally to not cause excessive load on the live database and to generally have faster access.

We have not used the whole document history of the IETF since its inception. The statistics below are all based on documents that have reached RFC status within the last 5 years (from March 2010 till March 2015). Only those recent documents

are being considered since the idea was to capture the current pace of the IETF. We have not done any statistics on documents that have not reached RFC status.

3.4.1 Data Inconsistencies

During our work, it turned out that a lot of the data in the database is not always accurate, some data seems to be missing and it is generally difficult to automatically process the data while making sure that results are indeed correct. We built in a number of cross-checks across database tables to filter out inaccuracies. The statistics here are based on the data that is left after applying these filters.

As an example, one problem with the data is that not all connections between documents are present. There are 473 RFCs in the considered dataset which do not have an individual draft in their document history, which seems very unlikely. After looking at some of those, we realized that the individual drafts themselves are in the database but the references to the respective RFCs are missing. We did not consider those documents for the analysis of the days it took to get from an individual draft to an RFC. We also filtered out 14 RFCs which did not have a working group draft connected to them. For three documents (RFC6353, RFC7396, and RFC7159) there is neither an individual nor a working group draft in the database.

After applying these filters, 1196 documents remained as part of our analysis.

3.4.2 Time-to-RFC

A very first basic question to ask is: how long does it usually take from the first individual submission of an Internet-draft till it becomes an RFC? This question is of course particularly interesting for researchers that would like to contribute research results to the IETF but also for researchers that consider a work-in-progress protocol draft for their research work.

Figure 3.1 shows the Empirical Cumulative Distribution Function (ECDF) of the duration from the first individual Internet-draft to becoming an RFC in days for all RFCs that were published in the last 5 years (time-to-RFC). It also shows the ECDF for the durations these drafts were individual documents and working group documents.

What can be seen in the figure is that the whole standardization process can take quite a bit of time. E.g. about 40% of all RFCs during the last 5 years took 3 years or more to produce. The top 10% even took over 6 years. At the other

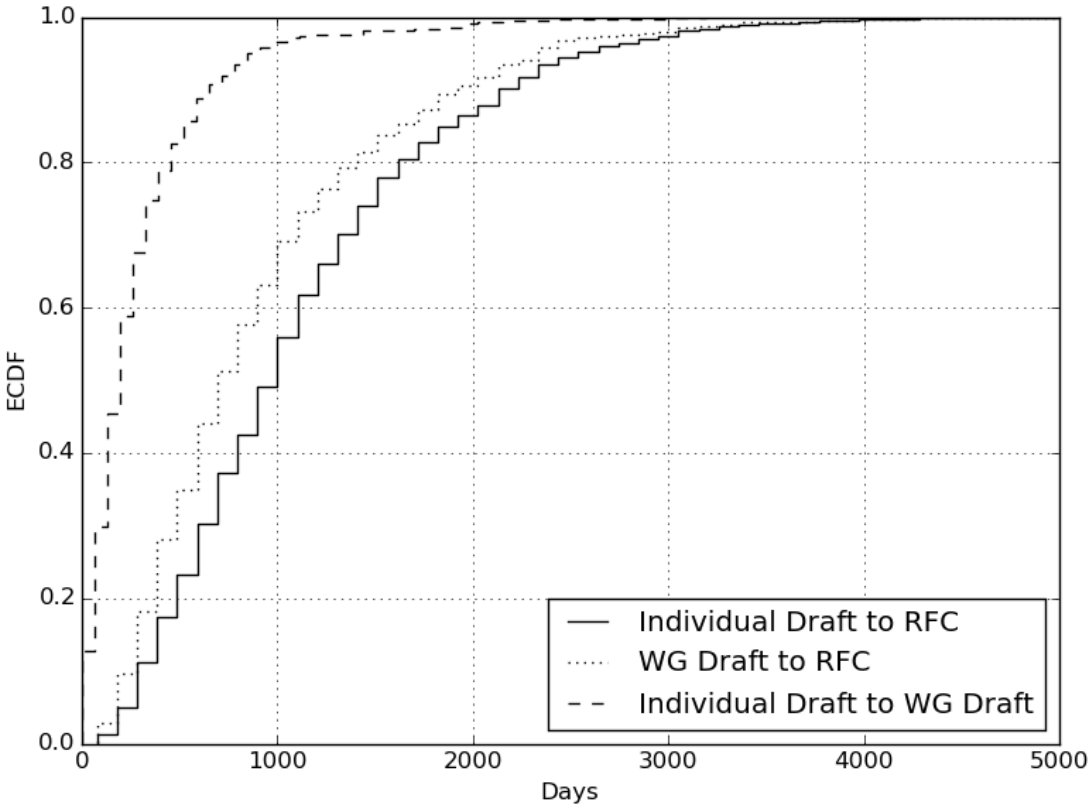


Figure 3.1: Time-to-RFC

end of the spectrum, 20% of all these RFCs took less than one and a half years to finish. This is a significant amount of time in particular for researchers who would like to contribute research work. They typically have already spent a considerably long time producing a research prototype before coming to the IETF. Given limited time and funding, this can be prohibitively long to see the whole standardization process through. But also for researchers that want to make use of work-in-progress protocols, these time frames are likely too long to keep up with changing protocol specs.

What can also be seen from the figure is that the time to WG adoption is much faster than the time to finish the working group process. Over 60% of all RFCs needed less than a year to being adopted by a working group. This is an interesting observation. If a document stays a considerable amount of time as an individual document, the chances rapidly decrease of ever being adopted and developed to become an RFC. The graph however also shows that there are a few RFCs that have spent many years as an individual document before entering the WG phase.

In other words, the time spent as an individual document is a not unreasonable indication for ultimate success of an Internet-draft to ever become an RFC. This in itself is not surprising, but the graph gives a good indication at which point success becomes unlikely.

3.4.3 Time Between Revisions

It is not just the overall time to finish a document that is interesting but also the time between revisions of an Internet-draft. This metric is basically an indicator of activity, progress and change and is of interest e.g. for early implementers of a work-in-progress protocol, as new revisions often translate to code changes.

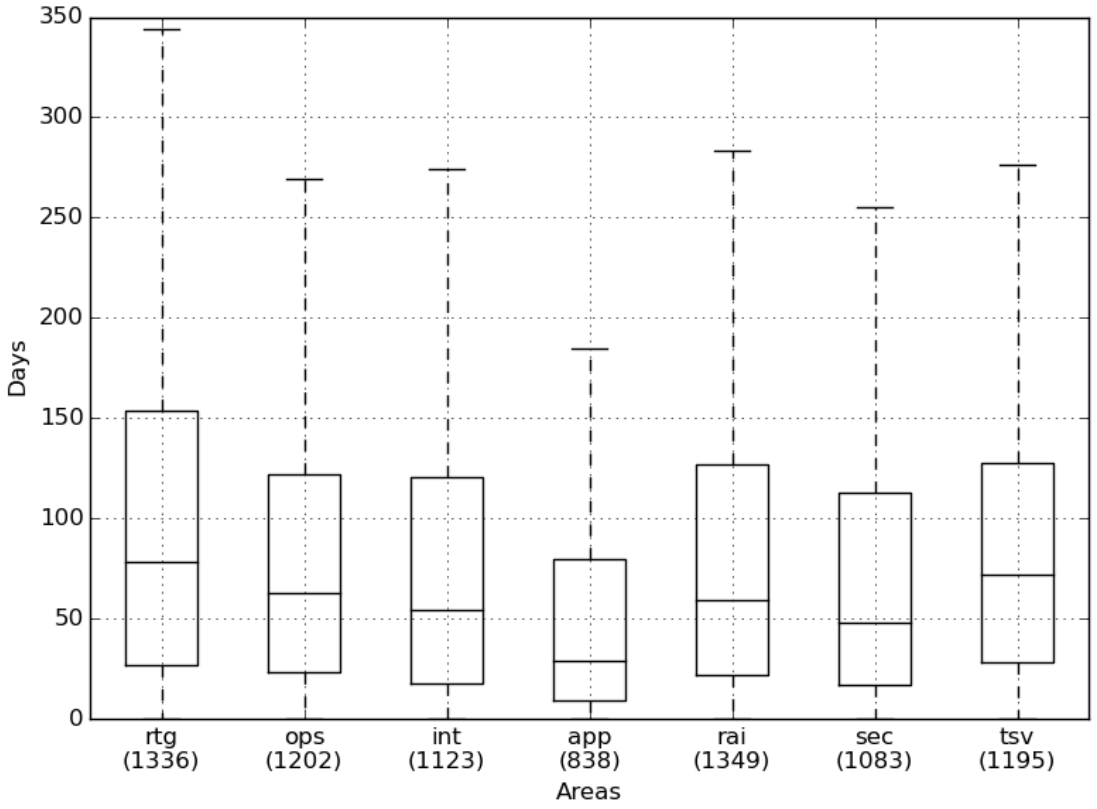


Figure 3.2: Days between revisions by area

We have looked at the average time between revisions and made statistics for the different areas of the IETF with the goal to not only learn about typical update cycles of documents but also to see whether some areas of the IETF perform "better" than others, where "better" is merely a reference to a higher update frequency and

has no implications on document quality.

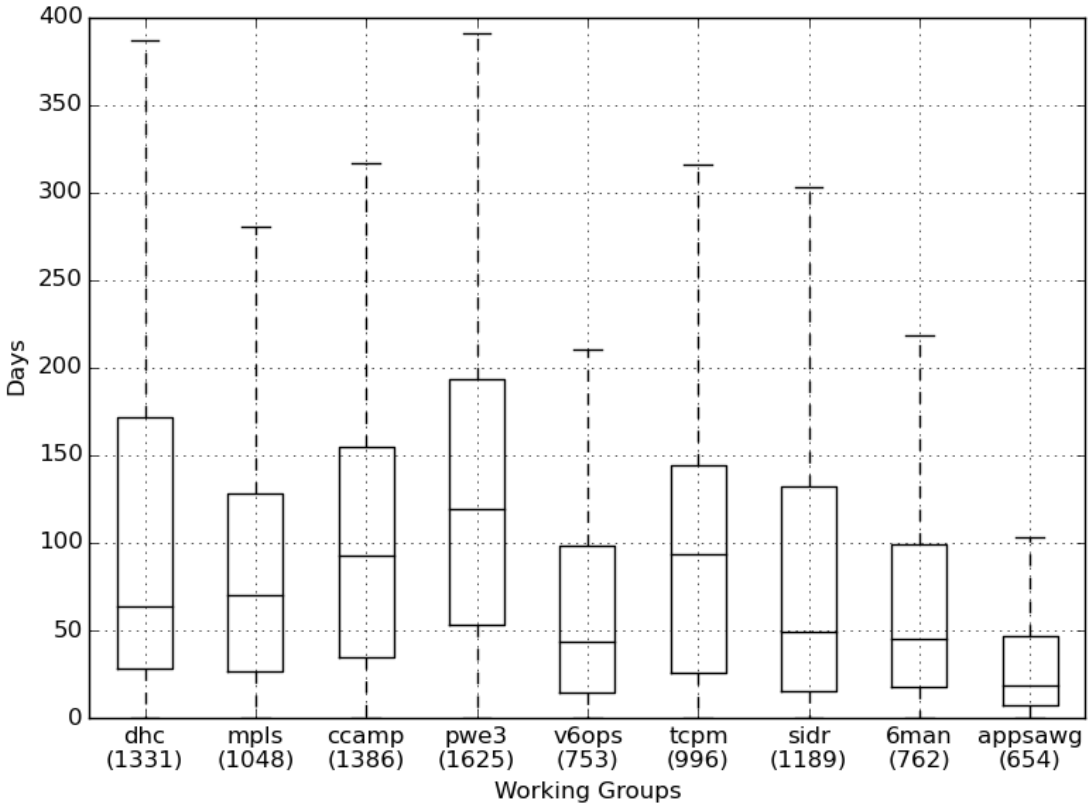


Figure 3.3: Days between revisions by working group

Figure 3.2 shows a box plot of the days between revisions per area. The boxes show the upper and lower quartile with the median in between while the whiskers show the lowest and highest values still within 1.5 interquartile range of the lower or upper quartile. Outliers are not shown but there are a few quite extreme outliers for some of the areas. As can be seen in the figure, the applications and security areas have faster update cycles compared to the other areas. Also, the variation across documents is notably less in these two areas.

The time between revisions itself can be a misleading metric however. E.g. it could be that certain groups have less revisions per document and can still produce RFCs faster. Therefore, we have plotted the average time to produce an RFC for that area below the x-axis. Even with that as a basis for a better comparison, the applications and security area are faster in producing RFCs whereas operations and routing need significantly more time.

Comparing areas as a whole, clearly glosses over the details of individual working

groups. Out of interest, we compared the working groups that produced more than 20 RFCs during the last 5 years, the result is shown in figure 3.3. It illustrates that also on the WG-level, there is a large dynamic range of document progress durations. E.g. the DHC working group (Internet area) and PWE3 working group (Routing area) both have comparably long update cycles, which is what the area statistics would have suggested. The 6MAN WG (Internet area) on the other hand updates documents much faster and the average time-to-RFC is considerably below that of its area. There is clearly significant difference between areas but also between WGs within an area.

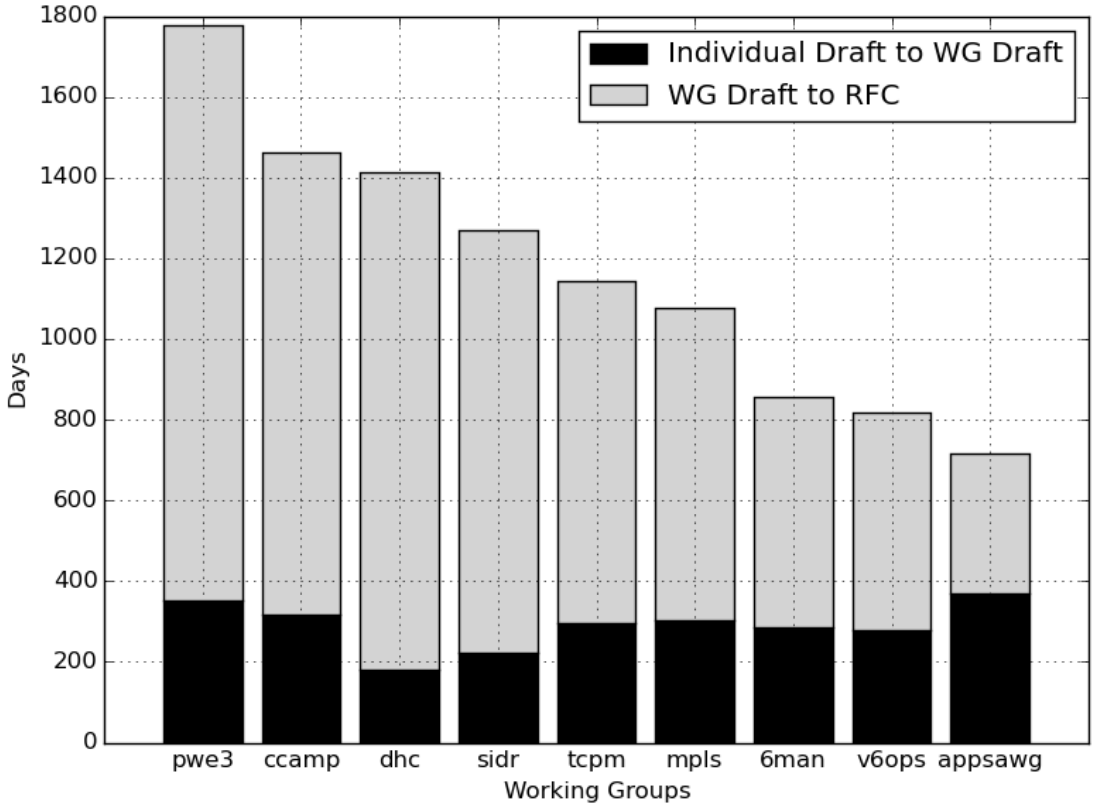


Figure 3.4: Time-to-RFC

Another question regarding these WGs is how much time is spent in which document phase, i.e. are there stark differences or are certain phases comparable across WGs. Figure 3.4 depicts the average time documents spend in these working groups as individual and as working group drafts. Interesting to note is that the time spent as an individual draft is comparable in a number of WGs, while there are large differences in the time it takes to complete the working group phase of a document.

There are two WGs that stand out. For one, the DHC working group very quickly adopts work, i.e. the time documents spend as an individual document is short, but once adopted, the documents need a very long time to pass the WG phase. On the other hand, the Applications Area Working Group (APPSAWG)—a WG that is very quick to produce RFCs overall—needs slightly more time to actually make a document a WG item than to progress that document to become an RFC.

3.4.4 Correlating Document Metrics and Time-to-RFC

Given an Internet-draft, it would be nice to have a metric that allows a researcher to estimate the remaining time that the draft needs to proceed to RFC status, i.e. the time to have a stable specification for implementation. The statistics so far only give a vague idea about certain aspects of the standardization process. E.g. it is safe to say that an individual draft still needs a lot of time to progress, since the working group phase typically is much longer than the time to being adopted by the WG, and that has not even happened yet. The statistics also give a general feeling for the overall time for documents to become an RFC, at least for the average case. It is however difficult to estimate the time to progress an Internet-draft to RFC status more precisely than that. It would be helpful if there was a simple metric that aided such an estimation.

An intuitive metric to estimate the time for a document to become an RFC would be the length of the document. If a document is long it would seem that the probability for that document to take long to become an RFC is high and vice versa. To see whether there is some truth in this intuition, we plotted the time-to-RFC against the page length of documents to see whether there indeed is a correlation. The result can be seen in figure 3.5.

Unfortunately, it turns out that this intuition is not a usable aid to estimate the time to finish an Internet standard. It really seems that any possible combination of pages vs. time-to-RFC exists.

Another attempt would be to use the RFC categorization as a basis to estimate the time-to-RFC. Figure 3.6 and figure 3.7 show the ECDF plots for informational RFCs and proposed standard RFCs—the two most common types. Two things can be seen from these two graphs. For one, the category is not useful as a hint towards the estimated standardization process duration. And also, independent of the category of the RFC, the distributions of the time-to-RFC are very similar. In other words, the category of a document does not clearly allow to make more precise estimations

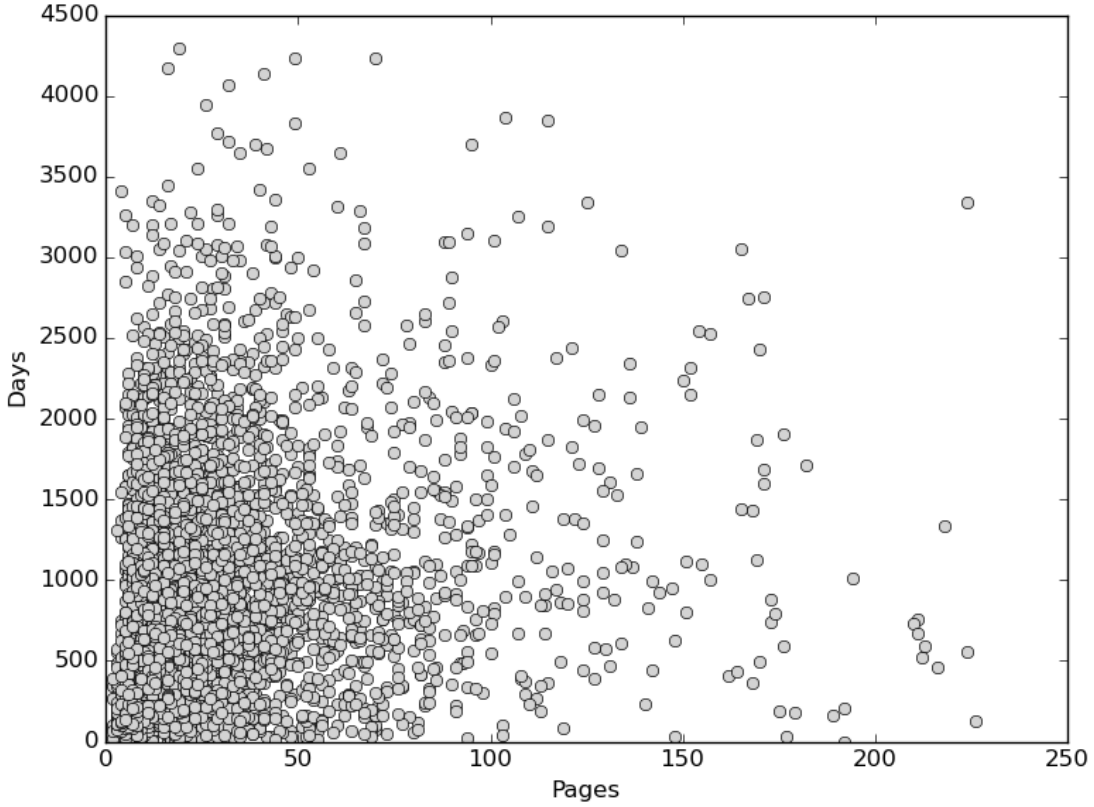


Figure 3.5: Number of pages vs. days to RFC

of the IETF process duration.

3.4.5 IETF Output

Finally, we looked at the yearly output of the IETF using all the available data in the datatracker database. We wanted to make sure that the magnitude of the IETF's work is understood and appreciated. So far, this text probably suggested the IETF to be a slowly operating organization that takes an overly long time to specify its protocols. It certainly does take its time to specify an Internet protocol, which is the price to pay for thoroughly designed things. But the IETF has been increasing its output year after year and is still producing high quality output. On top of all that, the organization is largely operated by people working on a voluntary basis. To give the reader a better idea about the magnitude of the IETF output and the underlying growth trend we plotted the number of documents per year since the late 1980s.

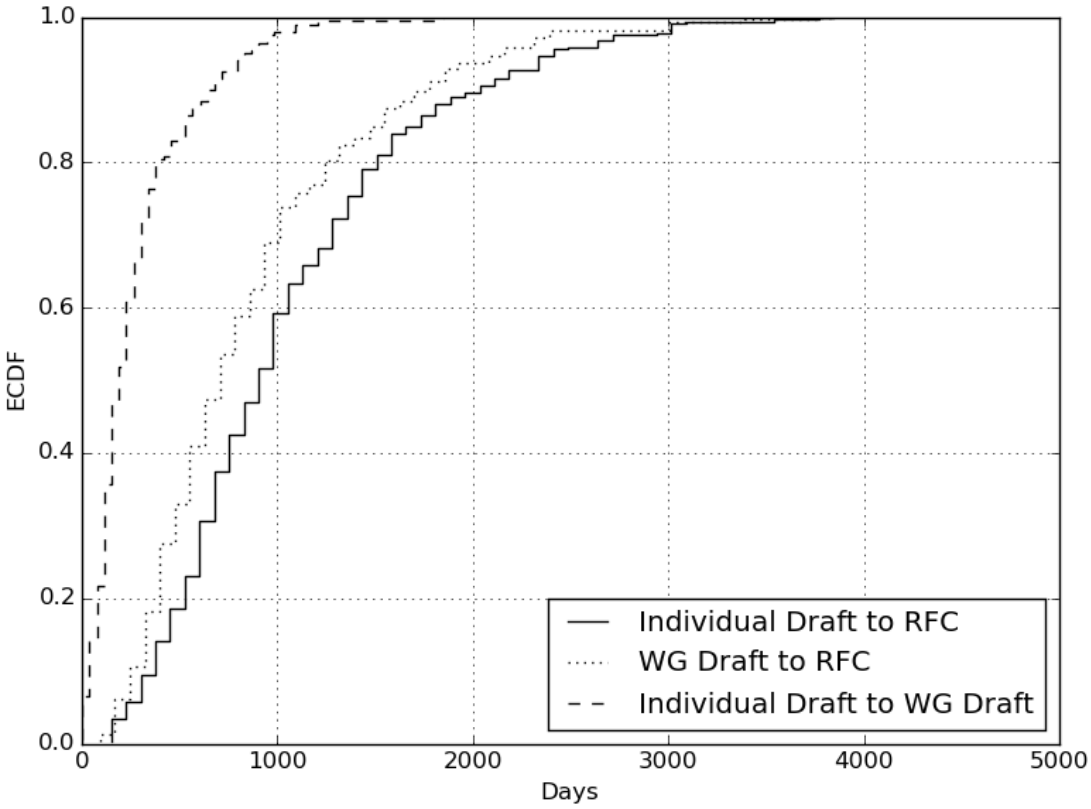


Figure 3.6: Time-to-RFC (Informational)

Figure 3.8 shows the published RFCs per year, but also the number of drafts adopted by working groups and the number of drafts submitted newly to the IETF as individual documents. There are currently about 1700 new documents submitted to the IETF each year, while there are currently more than 300 RFCs published per year. This shows that a large body of work submitted to the IETF never really goes anywhere. This work however is being dealt with, i.e. people review these documents, comment on mailing lists, present the work during the meetings etc. These are impressive numbers, given that this presents a significant amount of workload in addition to progressing existing work to RFC status.

3.5 Related Work

Fortunately, the IETF has a large number of participants that contribute code and other useful things to the community. Jari Arkko—current chair of the IETF—maintains a website with a collection of IETF process statistics at `http://www.`

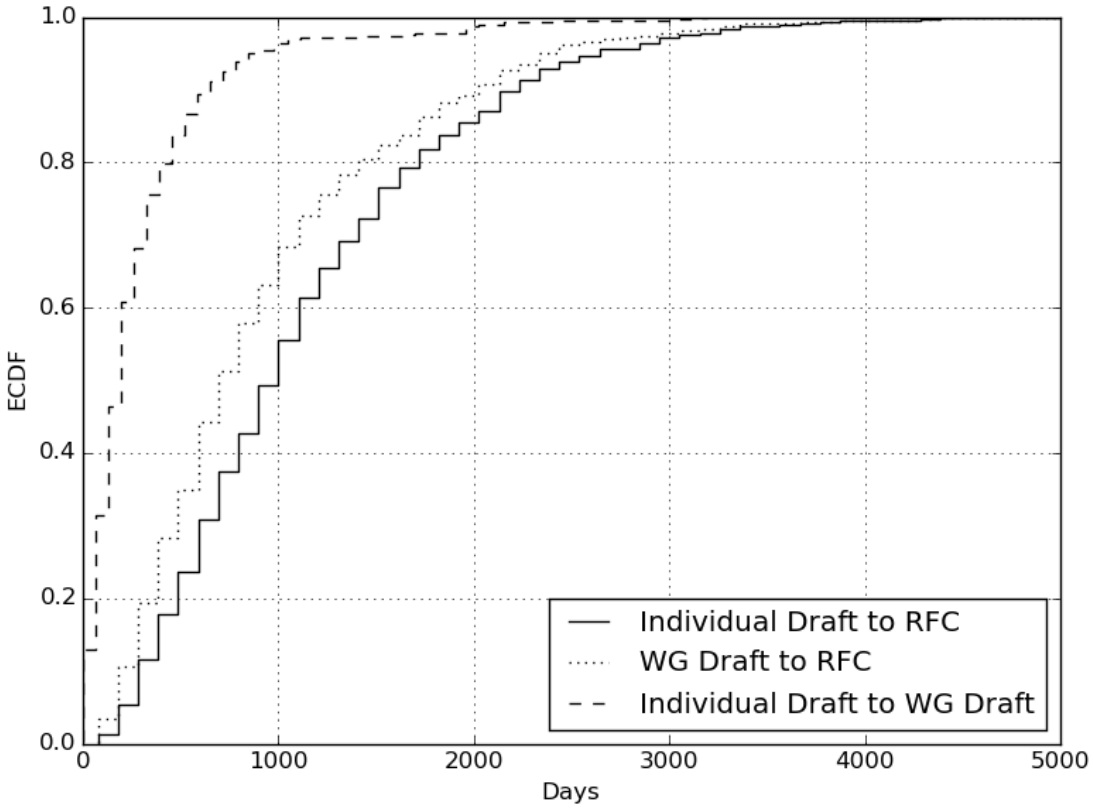


Figure 3.7: Time-to-RFC (Proposed Standard)

`arkko.com/tools/docstats.html`. The website contains a number of other statistics such as contributors per country or gender, individual document statistics or author affiliations. For this work, we focused more on timing properties of the standardization process and produced data that is not available on that website. We also used the datatracker data directly.

Additionally, there is an RFC on how to engage with the IETF as a researcher and what to expect when doing so. [EEBW11] describes the IETF process in more detail to the research community including difficulties a research-minded person might encounter when interacting with the IETF. The document should be the first read for researchers thinking about interacting with the IETF, it does however not give any recent insights into the time it takes to progress documents.

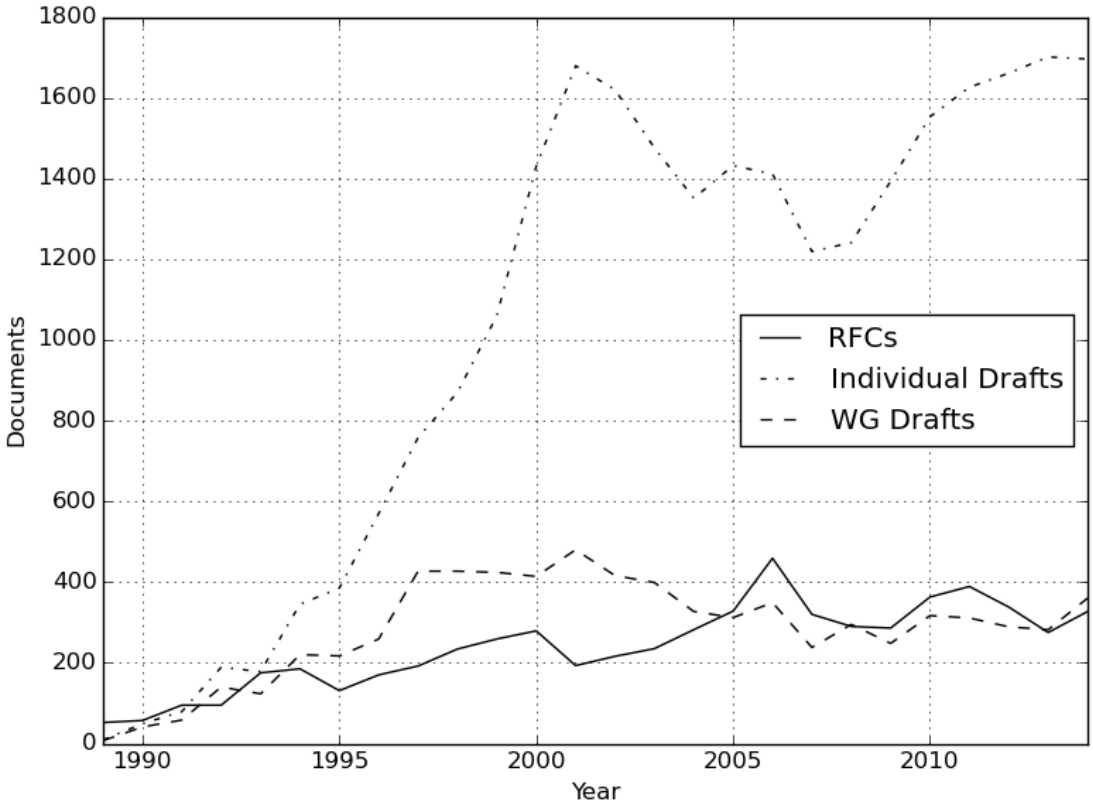


Figure 3.8: Documents per year

3.6 Conclusion

The analysis in this chapter has solely looked at the time it takes to produce an RFC. This is an important factor for researchers and early implementers that want to either engage with the IETF to specify a standard or that want to make use of a standard-to-be as part of their work, in particular if there has been no or little prior contact with the IETF.

Not unsurprisingly, it turns out that standardization is a time consuming process. Typically, the proponents of a technology spend years to create the final specification for an Internet protocol. We hope this work gives researchers a better feeling for the time it takes to produce standards documents in the IETF. The graphs shown are regularly generated and can be found at `http://net.hs-augsburg.de/ietf_stats/`.

3.7 Future Work

There are other useful metrics concerning IETF work which have not been addressed here. E.g. mailing list activity is a good indicator for the liveliness of working groups and the energy in a group to work on documents. Another useful, but difficult to estimate metric would be the amount of change per revision over time. Sometimes a document revision changes little more than the publication date, which is not a change that impacts implementations. Extracting this would however require manual inspection of all documents and certainly some expertise in the protocols being specified.

Some of the data shown should be put into perspective. We have analyzed different working groups and the time it takes them to publish RFCs. Some of these groups appear to be rather slow on average. E.g. the MPLS working group is one of the slower groups. The mailing list of this group however is very active. Also, the amount of output is impressive with already over 100 RFCs published. Additionally, the MPLS WG often closely collaborates with other SDOs such as the ITU-T, something a lot of the WGs do not need and therefore can move faster. We have not analyzed this for all the WGs under observation since for our general goal the underlying reasons were not that important and generally difficult to extract in detail.

3.8 Acknowledgment

The authors would like to thank Henrik Levkowetz for explaining some of the data inconsistencies and helping with access to the database in the first place.

The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 318627 (Integrated Project “mPlane”).

4 Measurements with the Masses

This chapter mirrors a paper published at the IRTF & ISOC Research and Applications of Internet Measurements (RAIM) Workshop in 2015 [FW15]. It describes the challenges encountered while developing and using a measurement platform which is intended to run on the devices of end-users, like GLIMPSE. Apart from the purely technical challenges which in themselves are often non-trivial—such as software updates, crash reporting and analysis, Network Address Translation (NAT) traversal and others—the non-technical challenges play a big role as users need to be convinced to install the GLIMPSE probe in the first place. The following sections describe these challenges in detail.

4.1 Abstract

The network measurement community has produced an impressive set of network measurement tools to which a number are added each year. A few prominent examples have seen wide-spread deployment covering large parts of the Internet but there is arguably one network segment which is extremely difficult to cover: host-to-host measurements with measurement instrumentation on "regular" end-devices. The problems in this space are manifold, but the main challenge is non-technical: incentivizing "regular" Internet citizens to take part in an experiment. In this chapter, we look at the challenges, examples of measurements which involved end-users and conclude with potential actions which would make measurements involving end-users easier (but by no means easy).

4.2 Introduction

A large set of network experiments have been carried out in the past on networks which are open for experimentation, where a large set of end-devices are under the control of an experimenter but which are neither attached to "regular" Internet

access networks nor are these devices "common" end-user devices. A prominent example of such a network is PlanetLab[1]. For many experiments such a network is not only sufficient but ideal because of the tight control and root access on these devices. For other experiments, these environments are not suitable because the Internet experience for real Internet users cannot adequately be captured.

There are many factors which influence the user-perceived Internet quality and which therefore also influences network measurements. For example the type of connection used (like WiFi, 3G/4G, DSL or cable) or cross traffic generated by other users in e.g. a home network or applications generating traffic in the background (like updates from an application or the system) influence a user's experience. Also the workload and system limitations of the end-device, or the workload of the requested service or the used path in the Internet all influence the end-users' Internet experience, albeit not all of these factors are actually due to characteristics related to the network.

These conditions of course can be simulated in a controlled environment once their parameters are known, but having an experimentation platform with real end-users allows to analyze the actual end-user experience with all foreseen and unforeseen problems. It also allows to assess how the Internet performs from a regular Internet user's point of view and measurements include all potential network segments starting from the home and access network. On the other hand, involving real end-users in network measurements is challenging to say the least and the reasons are manifold —many of which are addressed in the following.

4.3 Challenges

The following list of challenges is certainly not complete nor is the ordering to be understood as a prioritization. Also, for some measurements a certain challenge might not apply.

4.3.1 Access-Rights on End-Devices

A number of measurement tools need super-user access on an end-device in order to have access to low-level system functionality. This is difficult to explain to regular end-users since it would potentially allow an application to see all traffic sent by a device and access all data and functions available on the device without further

[1] `https://www.planet-lab.org/`

consent, which is unusual for typical applications installed by end-users. Usually, the operating system issues a type of warning or notification to the end-user when an application requires such privileges. This would require immense trust in the application developer which cannot generally be expected.

In addition, on some mobile operating systems an app developer has to specify which capabilities are accessed and the user has to grant those when installing the app or once the app needs to use them. If a network measurement app for a mobile platform makes excessive use of the available capabilities for the sake of better measurement results, a user might be reluctant to install this app altogether. At least a detailed explanation of why a capability is needed has to be provided, but it is questionable if a user will even read such information. Therefore, the less privileges an application needs, the higher the chances an application will be installed by a user.

4.3.2 Cross-Platform Development

A large set of operating systems are being used by end-users. Of course there are certain mainstream platforms which would already cover a significant potential user-base, but many of these platforms cover a narrow device spectrum such as Android for example. Therefore, to not only increase the potential user-base but to also cover a larger set of device classes and potential network segments, a measurement tool should be developed cross-platform to the largest extent possible.

A simple answer seems to be to use a language like Java, but there are often drawbacks such as access to low-level (non-root) features that are available to languages such as C++ and low-level access is often required for network measurements. This aspiration to develop cross-platform however comes with significant development costs. It means that a GUI version needs to be developed with multiple versions of the GUI for different screen resolutions and input methods (e.g desktop vs. mobile). It also means that the measurement code might differ from platform to platform. For example Mac OS allows opening non-privileged ICMP sockets which can be used to implement a non-root ping which is not trivially possible on other platforms. In addition, certain APIs/frameworks/libraries like PCAP might not be available on all desired platforms. In summary, cross-platform development is a very costly and difficult process.

4.3.3 Incentivizing the User

A potential end-user of a network experimentation platform needs some form of motivation to participate. Such platforms can be divided coarsely into two groups: measurements as the main cause of the application and measurements as a by-product of the application.

In the first case, the motivation for a user can e.g. be to use the platform for troubleshooting a problem he or she is experiencing while using an Internet service. This has the interesting side-effect of executing measurements under conditions which a user considers as problematic and the application can try to find the root cause for the perceived problems and provide possible solutions to the user. Other users might use the platform to have a look at measurement results for their own device or network out of curiosity (e.g. performance comparisons of operators). This might only be of interest to users with a technical or scientific background, but it can be an important factor if the gathered data can be visualized in an appealing and informative or even educational fashion for non tech-savvy users.

In the second case—measurements as by-product—the user installs an application which offers some service or information other than measurements, for example a game. By using the app the user grants the permission to also do network measurements, therefore the experimenter incentivizes the participant by providing another benefit.

In both cases it can be helpful in terms of motivation to provide global statistics and insights gained with the experiments. These statistics can e.g. be used to "gamify" the application[KB13]. A number of games and platforms e.g. motivate users by providing a form of score-board by introducing a kind of points system or "achievements" depending on the rate of participation so he or she can compare himself or herself to other users.

End-user participation is arguably the most challenging part about deploying an end-user-based measurement application. There are successful projects that have involved a significant number of end-users participating in projects without providing a direct benefit, i.e. the users support the cause of the project alone. BOINC[2] or Folding@home[3] are prominent example of this, but replicating this for network measurements is certainly difficult since many of these projects have significant, humanitarian goals.

[2] `https://boinc.berkeley.edu/`
[3] `https://folding.stanford.edu/`

4.3.4 Potential Interference

Devices of end-users are obviously used and potentially heavily so. Therefore, the measurement application has to make sure that the measurement results are accurate and not an artifact of other factors such as device usage or system limitations or at least label the results accordingly. Other factors can be cross traffic generated by other users in the network, or other applications and services running on the same device either in the background or in the foreground, for example when the user is browsing on the Internet. But not only network utilization needs to be taken into consideration, it is also possible that a device has very limited processing power or memory and a measurement cannot be executed in time or at all because of that.

As described in [SOB+14] cross traffic in the local network can easily be detected if the Internet gateway device offers traffic counters for the WAN interface queryable by a uPnP request. Monitoring the processor and memory utilization on the other hand is more difficult if multiple platforms should be supported because the retrieval of such information is different on many operating systems.

Generally, the control over or the monitoring of the local network and device are important when the accuracy of measurement results are to be ensured. Therefore, additional instrumentation might be needed for a measurement application on end-devices.

4.3.5 Availability

End-user devices do not necessarily run all the time. They might be turned off, especially at night. Even if using dedicated probes like RIPE Atlas[4] deployed in a home network it might not be possible to execute measurements all the time. Some users turn off the Internet gateway device automatically at night or if not used for a longer period of time. If the device uses a mobile connection like WiFi or 3G/4G it is also possible that an Internet connection is unavailable for shorter periods of time. The measurement application should be prepared to be shut down unexpectedly and without warning, also measurement results—if they can be gathered at all—need to be cached until Internet connectivity is available again.

Another aspect which has to be taken into consideration is what happens if the network interface or type of connection changes while doing a measurement. This can be the case if the device is a mobile device like a smartphone or notebook and

[4] `https://atlas.ripe.net/`

switches between LAN, WiFi or 3G/4G. If a user uses a mobile data plan with limited traffic it is also important to inform the user about traffic consumption and offer an option to prevent measurements to be executed when using those connections. Also, if a series of measurements is executed before and after a connection has been throttled by an operator due to exceeding the data plan's traffic budget, the results might be affected by that.

Another problem that influences the availability of devices is the prevalence of NAT. When designing the measurement infrastructure, the presence of NAT needs to be taken into account. E.g. end-to-end measurements, i.e. measurements between two clients running the application can be difficult when both are behind a NAT and a NAT traversal mechanism should be provided.

Availability is a problem that affects many aspects of a measurement application and the whole platform used to coordinate measurements and to collect measurement results.

4.3.6 User Privacy

Very early in the process of developing a network measurement platform one has to carefully consider data privacy issues. In particular, this includes which data to gather, identifying, discarding or anonymizing personally identifiable information, and how to store results in a law-abiding way [AP07].

Many users might have very understandable resentments to install a measurement application because network measurements can be misinterpreted as some kind of surveillance—in particular after Snowden. This is another social, but even more so a legal issue. Also, legislation regarding data privacy is different across countries and therefore a good guiding principle is to collect as little information as possible and to inform the user well what kind of information is being collected and why. In particular, it is necessary to make it clear that no personally identifiable data is being gathered and to provide an explanation of the experiments for a non-technical audience.

Because legal issues are possible, a lawyer and data protection professional needs to be involved. An EULA needs to be carefully formulated which should be stable, because changing EULAs are not only an annoyance but (rightfully) a cause for suspicion by users. This ties down the measurement tool significantly and future extensions will be difficult. Therefore, this is a key step and often the most difficult for the experimenter itself.

4.3.7 Measurement Coordination

For certain measurements, in order to make them comparable, to achieve a certain order of measurements from different vantage points, or to make sure that end-devices do synchronized measurements, a coordination of the execution times is needed. User devices are likely not time-synchronized using e.g. an NTP server, so the offset to a known time reference should be calculated to store the correct execution time of a measurement (using UTC as timezone). A re-calculation of the offset needs to be done on a regular basis in case the measurement application runs over longer periods of time due to potential system clock drift.

As mentioned, coordination is also important to make sure that e.g. measuring against one central end-point does not overload this end-point due to synchronized measurements. For one, this influences measurement results, but this might also be identified as a distributed denial of service attack, which could lead to widespread filtering of the apps traffic. To handle this, measurements could for example be executed with a random offset on each device.

The need for coordination differs depending on the runtime characteristics of measurements: continuously, repeated, or one-shot. In the first case, a measurement runs as long as the measurement application is executed, for example a passive observation of network traffic or a continuous ping. To the second type belong measurements which are executed regularly, influenced by a schedule provided by the platform operator. The last case only executes measurements once, possibly triggered by an event (like the user telling the application to measure now or after automatically detecting a problem in the network). But many measurements will certainly need some form of coordination.

4.3.8 Operations

The goal of any measurement platform involving end-users is certainly good coverage, which includes geography, ASes, access network types, device types and demographics. This ultimately means that the operational system will eventually need to handle a large user base which translates to a lot of coordination, failure, updates, a huge amount of data including a huge amount of noise.

The potential scale of such a system is difficult to handle and difficult to operate as a research group both financially and in terms of developer resources. There is little return on investment as an academic on the tedious maintenance and operational

aspects of such a system, in particular given all the challenges described so far and the marginal outlook on success.

Success in that respect is a blessing and curse at the same time. Unless the application has a one-shot measurement nature, data retention, archiving and other data management task will increasingly become a problem and need to be planned well ahead. Updating the app, a kill-switch for network measurements and other vital operational mechanisms need to be thoroughly tested before going live.

4.4 Conclusion

There already have been a number of successful measurement platforms or applications involving end-users. Out of these Dasu[SOB+14] (see also references therein for a good overview of other tools and platforms) particularly stands out as it has solved the problem of incentivizing the end-user and has activated over 100,000 end-users as participants. Dasu can be installed as a stand-alone application (where support for the Dasu-cause and altruism are the main drivers for installing it) but it was first deployed as a plugin for a popular BitTorrent client leveraging the popularity and the deployed base of the application. Measurements here were a by-product, but Dasu also performed active measurements to characterize the user's ISP.

We faced many of the challenges above while building an end-user-based network measurement platform ourselves called GLIMPSE[5], which is currently deployed on embedded, dedicated probes, but is also available for mobile and desktop systems. The development process was challenging and very time consuming and the most challenging bit is still ahead—convincing end-users to install the app. The value proposition is simple: identifying potential network issues. Other than that, the app offers the implemented network measurements as built-in tools for the user to trigger measurements him or herself, targeting "advanced" users. That leaves the cause of the platform for the altruistic, cause-driven users: we build an application to "measure the Internet", where measuring the Internet means our measurement coordinator triggers measurements, or series of measurements to capture certain Internet-characteristics as part of a purpose-driven measurement campaign. A user basically donates a certain amount of bandwidth for these measurements per month.

The real question is whether there is something that can be done to make end-user based measurements easier, measurements more expressive and accelerate the

[5] `https://www.measure-it.net/`

development of measurement tools. We believe there are things that can be done, non of which are particularly easy but all well worth pursuing. We see these mainly in two areas: protocols and standards, libraries and APIs.

Available protocols and standards constrain what can be measured in many respects. It is quite difficult to do very trivial measurements today. E.g. measuring the return path from a destination back to the source typically requires control over the destination. Allowing more measurements to be taken is of course a careful consideration between the value a new measurement gives and the potential dangers a new measurement introduces such as potential misuse for Denial of Service attacks. There have been efforts in the past to introduce new protocols such as the IP Measurement Protocol (IPMP)[6] but introducing new measurement capabilities is quite difficult. Instead of introducing new protocols, it would be possible to extend protocols such as ICMP using new types but that as well has not happened recently. At least type ranges have been set aside for experimentation by RFC4727. In essence, standardizing protocols or protocol extensions for use in end-user measurements is very difficult to say the least but would massively help to gather a better understanding of the Internet and to better troubleshoot problems experienced by end-users. On the positive side, the IETF is in the process of developing a standard for measurement coordination in the LMAP working group, which could at least harmonize this aspect of network measurement coordination.

The second area really concerns the development cost and the time spent developing and testing the measurement application. The development burden could be significantly reduced if libraries would be actively maintained and contributed to for cross-platform development of network measurements. Many of the challenges described above resulted in significant effort working around platform-specific differences. But many of the measurements such as a simple UDP-based ping would certainly be useful in many network measurement contexts. Building up a library and agreeing on APIs would greatly speed up the development process and lower the development costs for future tools to come.

There is probably little that can be done about the operational complexity of a measurement platform involving end-users. Also, the legal side is something that every tool needs to cover individually and no simple recipe exists that is suitable for a wide range of network measurements and experiments. The two areas described above also will need broad consensus as these protocols or protocol extensions will

[6] `https://tools.ietf.org/html/draft-mcgregor-ipmp`

likely have to be implemented by operating system vendors and large support by developers and implementers. It is also worth asking whether certain OS features can be adopted by other operating systems. E.g. non-privileged ICMP sockets as provided by OS X are very handy and do not seem like a overly dangerous security hazard. But this—just as the other things mentioned— requires significant support, interest and time.

5 Implementation of the GLIMPSE Measurement Platform

The University of Applied Sciences Augsburg develops the end-host-based network measurement platform GLIMPSE (Global Internet Monitoring and Probing System). It is a system with central controlling components and distributed active measurement probes. A summary of GLIMPSE has already been published at the International Conference on Performance Evaluation Methodologies and Tools in 2015, this chapter here reflects the current state of the platform with a much more in-depth description of its components [FW16].

The original intend of GLIMPSE—which differs from the actual system implemented today—was to have a measurement platform where end-users where expected to download the probe for two reasons: Firstly, they should be able to trigger measurements when they realize something related to the Internet is not working properly in their network or on their device. For this reason the app should have three buttons.

1. My Internet is not working
2. My Internet is slow
3. Run a speedtest

The first two buttons should include a process to determine what the user is experiencing by asking questions like "What is not working? Is it the Internet as a whole or just a service or a website? Which website?" and simultaneously perform measurements in the background. This should lead to more precise measurements to determine the cause of the problem. These measurements are of great interest for researchers as regular measurements mostly measure the cases when everything is working as expected. By relying on the user to trigger measurements when they see the need for it, measurements should bring up results for interesting circumstances when something is not working properly in the network or at a service provider. The

third button should just run some static measurements like a latency measurement and a speedtest to determine the current performance of the connection not only in the case of failures.

The second reason for end-users to download and install GLIMPSE was based on potential altruistic motivations of end-users. By installing the app, users could donate a specific amount of traffic volume which researchers then use to perform measurements from the users Internet connection for centrally-specified measurement campaigns.

Some months after the development had started it became obvious that those goals were too ambitious given the project's lifetime and number of persons involved. Specifying and implementing a measurement platform for end-users and on end-users' devices like smartphones is in itself a large task even without involving (semi-) automatic reasoning on measurement data. Also, designing an appealing user interface is hard when the main attention and expertise of the developers is in writing network measurements. Therefore, the focus of the project shifted to first implement a running and well-tested platform which implements the following (for a measurement platform typical) work flow:

1. Specification of a measurement instruction on the probe or in the back-end
2. Execution of this instruction on one or multiple probes
3. Sending the measurement results to the back-end to process them there (by storing and/or analyzing them)

These and other things were done within the lifetime of the project, always with the original intend of GLIMPSE in mind to allow an automatic reasoning of end-users' problems in future versions.

Details on the adapted design goals, an overview of the architecture with back-end and probe as well as a description of the implemented measurements can be found in the following sections.

5.1 Design Goals

While the initial focus of the project changed, most of the design goals for GLIMPSE remained unchanged throughout the development. They are described here together with some high-level descriptions on how some features were implemented.

Software-only Platform

To be independent of specific hardware, GLIMPSE is a software-only solution. It is possible to execute it on single-board computers and is in fact distributed on BeagleBones and Raspberry Pis by the project maintainer to other researchers, similar to the RIPE Atlas project. But the source code of the software is available under the BSD license and can be compiled and executed on any supported platform.

Cross-Platform

To make the probe available to as many users as possible and be able to measure the same things a user would experience for example when using a smartphone, a major design goal of GLIMPSE is being cross-platform. The probe is written in C++ using Qt [The], a framework for (besides other things) cross-platform development. This allows to compile and run GLIMPSE on any platform supporting Qt, which includes Microsoft Windows, Linux-based systems, Mac OSX, Android and iOS. Under some circumstances, platform-specific implementations were needed. In these cases, the project focused on the previously listed platforms.

Target Audience

All parts of GLIMPSE were developed with the end-user as target audience in mind. Therefore, the app is primarily run on end-user devices with other processes running simultaneously. This means for one thing, that all operations have to have a small resource footprint and should not introduce heavy load onto the system. Ideally, the user and other applications should not be influenced by the measurements executed in the background. But this design goal also has to be kept in mind when designing measurement methods. They in turn could be influenced by other applications running on the same host, for example by system load or memory consumption. Additionally, network traffic within the user's network (cross traffic) can have an impact on measurement results.

Access Privileges

As GLIMPSE should be installed and used by end-users, the barriers to do so should be as low as possible. Convincing users to install an application which requires root privileges to perform network measurements on a private device is problematic as this introduces multiple potential security and privacy risks. On some devices it is

not even possible to obtain these privileges without modifying the system's core. Prominent examples are the Android and iOS platforms where it is not possible to execute software as the root user. Introducing this design goal somewhat limits the power of GLIMPSE but also limits the security risks when using the software and potentially increases the user base.

Scalability

To be able to do larges-scale measurement campaigns, the platform should be able to dynamically allow new probes to connect to the system, distribute configurations and measurements automatically and gather results centrally in a database.

Small and Simple Measurements

GLIMPSE should provide a set of tools to execute simple and reliable measurements. One measurement itself should be a small and fast way to determine a meaningful network metric or condition on the host, in the hosts network or beyond that. But a combination of these tools should allow an experimenter to specify complex measurement campaigns. Future plans for GLIMPSE include the definition of a language to chain measurements to influence the used tools and parameters by the result of a previously executed measurement.

5.2 Architecture

GLIMPSE uses a pull architecture from the user's perspective, meaning all connections are initiated from the probes. This is necessary, as most probes are in the home network of users behind a NAT gateway and are therefore not directly reachable from the Internet. In the following, the back-end components as well as the probe itself are described with their functionality and connections to each other. An overview of the architecture of GLIMPSE can be found in figure 5.1.

5.2.1 Supervisor

The central point of control for the distributed probes is the supervisor. Every probe has a pre-configured supervisor which is used to pull its configuration on the first start of the probe. A snippet of the currently used configuration can be seen in listing 5.1. This configuration is obtained periodically to check if a new supervisor

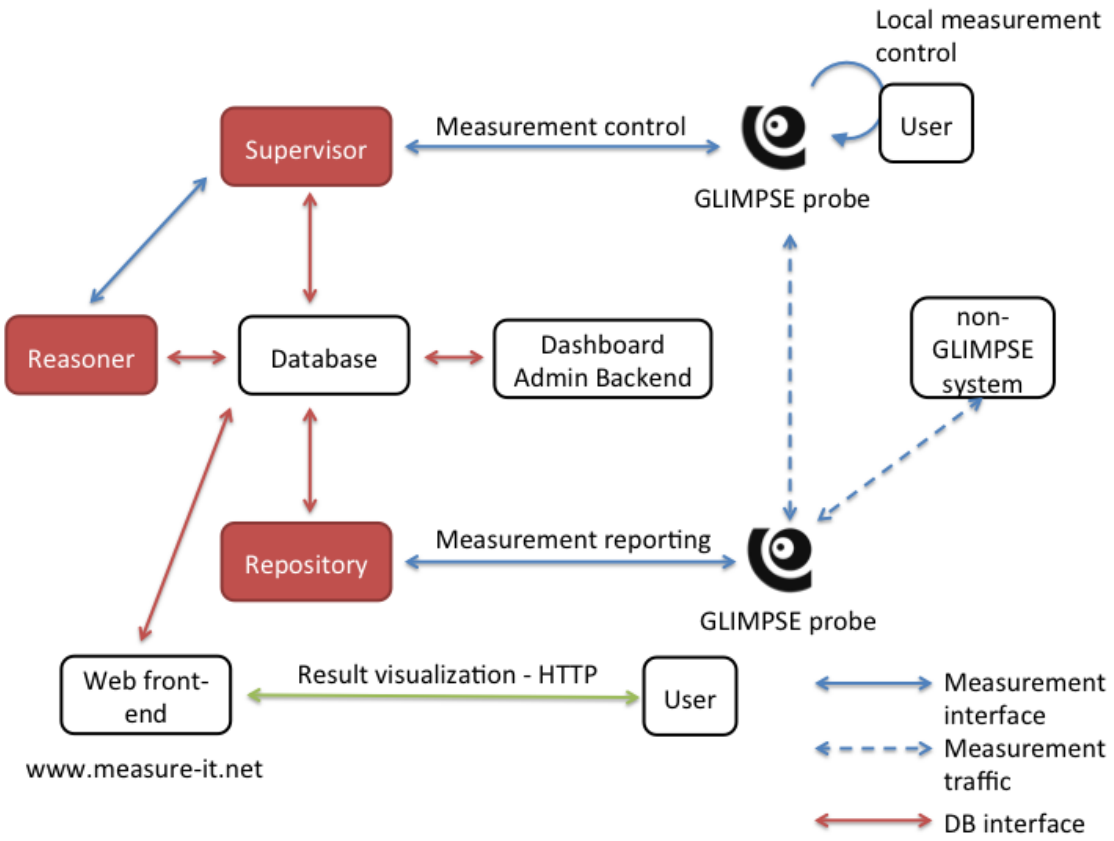


Figure 5.1: GLIMPSE architecture

or repository should be used or if the interval in which to pull the configuration has changed.

Temporal Scope

The temporal scoping (timings) used here is the same as for all the other components of GLIMPSE. As GLIMPSE was developed while the LMAP working group was still in the process of specifying the timings, the ones used here are not exactly the same as LMAP uses today. They represent what was needed at the time while still trying to implement the current state of the art.

The **Immediate** timing is used to execute something as soon as possible. It takes no additional parameters and is equivalent to the `ma-immediate` object in the LMAP information model and the keyword `now` in the mPlane temporal scope.

To express a specific point in time the **OneOff** timing is used. It consists of a datetime parameter only and is equivalent to the `ma-oneoff` object in LMAP and a single time-stamp in mPlane.

```
"supervisor_channel": {
  "target": "supervisor.measure-it.net",
  "timing": {
    "periodic": {
      "end": "2099-07-16T13:15:09",
      "interval": 600000,
      "randomSpread": 30000,
      "start": "2014-07-16T00:58:00"
    }
  }
},
"report_channel": {
  "target": "repository.measure-it.net",
  "timing": {
    "periodic": {
      [...]
    }
  }
},
[...]
```

Listing 5.1: GLIMPSE configuration

The **Periodic** timing is the one which is used most of the time in GLIMPSE as it allows to repeat the execution of a measurement or the retrieval of information (like the configuration) by specifying an interval. Its parameters additionally allow to set a start and end datetime object, which specify between which points in time the repetition should take place. If those two are not set, the timing starts as soon as possible and reruns forever. To prevent for example multiple probes executing the same measurement instruction at the exact same time, a random spread parameter can be set. This adds a delay to the timing randomly selected within this range by everyone who processes this timing. If not used, a measurement from a large number of probes to a single target might overload it or trigger a system which detects Distributed Denial of Service (DDoS) attacks. Everything described for the periodic timing up until now is equivalent to the `ma-periodic` object in LMAP. In fact, LMAP gives the possibility to define a random spread for all types of timings, which was not implemented by GLIMPSE. mPlane on the other hand also allows a relative time interval as end datetime (for example to run a measurement for one hour), but has no random spread parameter. While developing GLIMPSE it became obvious that the periodic timing of LMAP and mPlane were fundamentally different and

could not express the same temporal scopes. This difference, which was the source of multiple discussions and side meetings at LMAP working group and mPlane plenary meetings, can be explained best with an example. A latency measurement usually consists of not only one but multiple ping packets sent to the same destination. A concrete measurement instruction could be "ping `measure-it.net` with four ping packets as soon as possible, the time between each packet should be one second". There are two ways to implement this:

1. Make a latency measurement with the parameter `measure-it.net` as target and use a periodic timing which starts as soon as possible, repeats four times and ends after that.

2. Make a latency measurement with the parameter `measure-it.net` as target, one second as the interval between packets and four as the packet count; use a immediate timing to execute this measurement as soon as possible.

When looking at this example the first solution might seem a more natural way to express the measurement. But what if this measurement should be repeated every hour? A typical scenario for a measurement platform is to monitor the availability of servers by doing exactly this— pinging every specified time interval using multiple packets. In the second implementation, the one that LMAP chose, one could just change the timing from immediate to periodic with an one hour interval. The first implementation did not allow this scenario at first, this is why the temporal scope was extended by the author of this document to support so called "repeating measurements" [TMF+15b, Page 26] by allowing to give an additional interval to repeat the original timing.

The **Calendar** timing is used to express more complex repetitions than just a fixed interval. Similar to a cronjob, a task scheduler in Linux/Unix-based operating systems, it allows to specify the month, day of the month, day of the week, hour, minute and—which is not included in a cronjob—even the second in which to execute something. Like the periodic timing, it also has a start and end datetime to determine a time frame for which to use it. The LMAP information model also specifies this exact same timing, while mPlane did not support it at first. But, as for the periodic timing before, this was also integrated into the mPlane architecture and specification by the author of this thesis with the same features LMAP defines.

A timing unique to GLIMPSE and not covered by LMAP and mPlane is the **OnDemand** timing, as it is kind of unique to the use case of GLIMPSE and not

needed by all measurement platforms in general. This timing is used, as the name suggests, not to execute a measurement at a specific point in time but to allow a probe to decide for itself to execute a measurement between an optional start and end datetime. GLIMPSE integrated this because often end-users are the first to recognize problems in the network and want to trigger measurements on their own. To have some control over the measurements executed and the parameters used, the app has some default measurement instructions with the on-demand timing set. Of course users can still execute own measurements (as seen later), but results of those are not sent back to the repository for privacy reasons and because it is hard to compare measurements when the parameters are not fixed. Providing some basic measurements with the on-demand timing allows to analyze the results of such measurements which might be used by end-users in circumstances of great interest—a moment when there are actual problems in the network—to researchers.

All timings and datetime objects in GLIMPSE use UTC as time standard. Table 5.1 summarizes the available types of timings with a description, examples and the respective mPlane representations.

Measurement Instructions

The configuration also provides a timing which tells the probe when to pull measurement instructions from the supervisor. Probes are typically on the premises of the user or installed on a mobile device. In particular the latter often have a volume-limited Internet plan and frequent polling of large instructions should be avoided. Therefore, the supervisor responds with a list of IDs for all available measurement instructions instead of the complete instructions themselves. The probes then request those instructions they don't already have downloaded and locally available. This reduces the traffic immensely as each probe pulls the configuration every 10 minutes in the standard configuration. To disable a measurement even before the end of its temporal scope is reached, the supervisor also provides a list of measurement instruction IDs to be delete from the probe. Modifying a measurement instruction is not possible as this would make results for the same instruction incomparable.

An example for a measurement instruction can be seen in listing 5.2. It contains the measurement method, the parameters (called options here) and the timing when (and how often) to execute the measurement. In this case the latency measurement is used to determine the latency to `www.measure-it.net` with four packets using UDP. The timing specifies the execution to take place every hour (the interval has

```
"id": 43,
"task": {
    "id": 14,
    "method": "ping",
    "options": {
        "host": "www.measure-it.net",
        "count": 4,
        "type": "Udp"
    }
},
"timing": {
    "periodic": {
        "interval": 3600000,
        "randomSpread": 30000
    }
},
"precondition": {
    "on_cellular": false,
    "on_wire": true,
    "on_wireless": false
}
```

Listing 5.2: GLIMPSE measurement instruction

Timing	Description	Example(s)	mPlane representation(s)
Immediate	Once as soon as possible	-	`now`
OneOff	Once at a specific point in time	13th of July 2016 at 6 am	`2015-07-02 06:00:00`
Periodic	Repeated between two specific points in time based on a given period	Every hour for the next 24 hours starting as soon as possible	`now + 24h / 1h`
		Every hour for one specific day	`2015-07-02 06:00:00 ... 2015-07-03 06:00:00 / 1h`
Calendar	Complex execution pattern based on a cron-like calendar format	Every first Monday in a month	`repeat now ... future cron 0 0 * 1,2,3,4,5,6,7 1 * now`
OnDemand	Used for measurement tasks which can be executed by the user whenever they feel the need for it. This type is neither specified by LMAP nor mPlane.	-	-

Table 5.1: Timings used in GLIMPSE

to be given in milliseconds) and the execution is delayed randomly by no more than 30 seconds. As shown later the ping method supports many more parameters, if they are not provided as part of the instruction, then hard-coded default values are used.

The last part of the instruction shows something GLIMPSE calls **preconditions**. These are conditions which have to be met on the probe for the measurement to take place and are evaluated every time right before the measurement should be executed. In this example the ping should only be run if the probe uses a wired connection, cellular and wireless connections should not be used. The other available preconditions not shown here are the remaining battery capacity and the device's location. In the first case measurement only takes place if the devices battery level is at least the given amount. The latter makes it possible to specify coordinates and a maximum radius, only probes within this area should execute the measurement. This precondition works only on Android and iOS deployments of GLIMPSE as the location can be determined from the appropriate APIs on these systems (if at all).

In contrast to preconditions which are evaluated on the device prior to the execution of a measurement, **qualifications** are used in GLIMPSE to deliver measurement instructions only to certain devices. The following constraints are implemented at the moment:

Operating System Allows to restrict instructions to probes running on certain platforms. The available values are: Android, iOS, Windows, Linux, Mac.

Country, Autonomous System Number (ASN) Based on the last known IP address of the device the country and the ASN can be determined and used to restrict measurements regionally and to certain ISPs.

Users, Devices, Groups Only deliver the measurement instruction to specific users (with all their devices), only to specific devices or to a specific group. The latter is used for a not yet released feature which allows users to join multiple groups which provide measurement campaigns with a certain subject or from a specific research group.

Qualifications are evaluated every time a devices tries to pull measurement instructions. The country and ASN restrictions are only meant to be used for short measurements, as every probe which pulled the instructions once is going to execute them as long as the timing specifies, even when the conditions are no longer met (e.g. when switching from the WiFi interface to a mobile connection and therefore changing the ASN).

5.2.2 Repository

As the architecture and configuration shows, the reporting of measurement results is done against another server: the repository. It might be (and it is at the moment) collocated with the supervisor but it is a separate piece of software. It receives, validates and stores the results in the central database, which is also used by all other back-end components.

The probes periodically (depending on their configuration) submit all their available results in so called reports. Each report includes multiple results, the time when the report was generated, the version of the probe and the measurement task it belongs to. It is important to note that a measurement task is different from a measurement instruction. The latter consists of exactly one measurement task as well as the timing when to execute it. It is not of concern for the results when it was planned to be executed, therefore only a reference to the task (and therefore the parameters) that was the basis for this result is stored. This makes it possible to compare results from the same measurement task but different measurement instructions, as often different measurement campaigns use the same tasks. An example is the previously shown ping task, which might be done on an hourly basis but also in another instruction only once when a problem was recognized. All results from this task can be in one report.

An example for a measurement report with one result produced from the previously shown latency measurement instruction is shown in listing 5.3. A full example for the results of a latency measurement (shown here truncated in the `probe_result` part) can be found later in this chapter. Additionally to the actual outcome of a measurement, some other information is part of each result: the start and end datetimes, the pre-calculated duration in milliseconds, an error-string if something went wrong, a UUID (which is explained later) and multiple pieces of information about the system the probe ran on taken directly before (`pre_info`) and after (`post_info`) the measurement. These include for example the system load (`cpu_usage`) to determine if the measurement was influenced by other running processes and some traffic counters recording the traffic volume GLIMPSE introduces on the probe (`used_traffic`). Not every value is available on all systems, it is for example not possible to obtain the signal strength (`signal_strength`) of the WiFi on iOS (at the time of this writing).

```
"device_id": "..."
"reports": [ {
  "task_id": 14,
  "report_time": "2016-07-14T20:58:09Z",
  "app_version": "2.7.2"
  "results": [ {
      "probe_result": {
        "destination_ip": "74.125.206.105",
        "round_trip_avg": 21.238000869750977,
        "round_trip_ms": [
          ...
        ],
        ...
      },
      "start_time": "2016-07-14T20:58:08Z",
      "end_time": "2016-07-14T20:58:09Z",
      "duration": 885,
      "error": "",
      "measure_uuid": "00000000-0000-0000-0000-000000000000",
      "pre_info": {
        "used_mobile_traffic": 0,
        "free_memory": 0,
        "battery_level": 55,
        "tbm_active": false,
        "connection_mode": 4,
        "used_traffic": 5716,
        "cpu_usage": 0.014084506779909134,
        "mm_active": false,
        "available_traffic": 524288000,
        "available_mobile_traffic": 31457280,
        "signal_strength": 68
      },
      "post_info": {
        "used_mobile_traffic": 0,
        "free_memory": 0,
        "battery_level": 55,
        "tbm_active": false,
        "connection_mode": 4,
        "used_traffic": 5716,
        "cpu_usage": 0.02857142873108387,
        "mm_active": false,
        "available_traffic": 524288000,
        "available_mobile_traffic": 31457280,
        "signal_strength": 69
      }
...
```

Listing 5.3: GLIMPSE measurement report

5.2.3 Reasoner

In general, a reasoner for GLIMPSE tries to automatically schedule measurements based on an initial setup and results from previously executed measurements. This could be used to drill down into problems or to run extensive measurement campaigns with multiple different measurements.

Only one reasoner module is implemented in GLIMPSE at the moment, the traceroute-reasoner. Similar to paris traceroute [AFT11], this reasoner tries to determine multiple paths between two end-hosts by modifying the flow identifier (using varying port numbers) of the packets used. The normal traceroute measurement implemented in many operating systems send multiple packets (usually three per hop) with an increasing Time-to-Live (TTL) value to a target. As every hop on the path decreases the TTL by one and only packets with positive TTLs are forwarded, every hop on the path should answer with an ICMP time exceeded packet [Pos81]. Normally, packets with the same flow identifiers are routed through the same hops to the destination. By modifying the source and destination port of the probing packets[1] this reasoner tries actively to find multiple paths to an end-host.

Technically, an experimenter chooses one or multiple probes to participate in this reasoning and automatically schedules ten traceroute measurement tasks with random source and destination ports for a given destination to be executed as soon as possible. Each time the repository stores a new result for these measurement tasks, it calls the traceroute reasoner that determines if a new, previously unknown path was discovered. If so, this path is stored and ten new traceroute measurements with random source and destination ports are scheduled. This continues until the results yield to no new paths for this destination.

This reasoner was implemented to have a working proof of concept for the reasoner interface of GLIMPSE. It remains future work to evaluate the results of the traceroute-reasoner, improve it and write other reasoners.

5.2.4 Dashboard

To give experimenters an interface for administrating and maintaining the system, a web-based Dashboard is used. It provides an overview of all available data including configurations, devices and tasks as well as visualizations for the results.

The screenshot in figure 5.2 shows an overview of the dashboard with some statis-

[1] The traceroute measurement method of GLIMPSE uses UDP packets as shown later

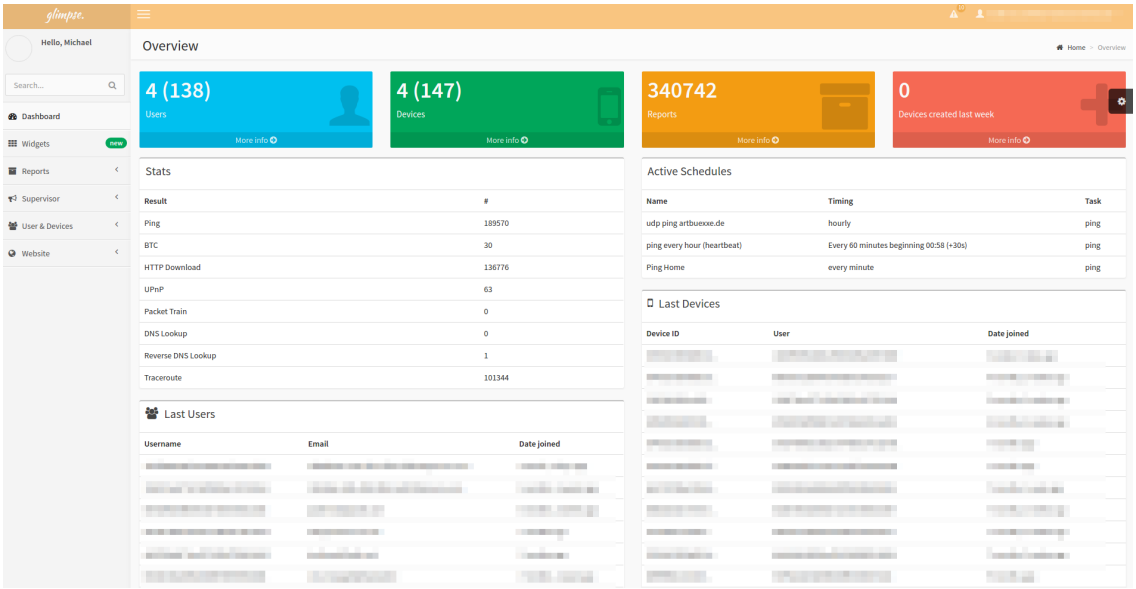


Figure 5.2: Overview in the GLIMPSE Dashboard

tics about registered users and devices (all data containing sensitive data has been blurred out). Numbers in brackets show the overall counts of users and devices, the first numbers are the values for the last 24 hours. As GLIMPSE is intended to be installed by end-users, every user has to register himself as well as the device before using the system. It is possible to register multiple devices per user, which later on allows users to see all their owned devices and schedule measurements for only them. Registration is normally done through the app, but the dashboard also allows to manually add new users and devices for debugging.

The main purpose of the dashboard is to create measurement tasks, timings and instructions and to view results for one or multiple devices.

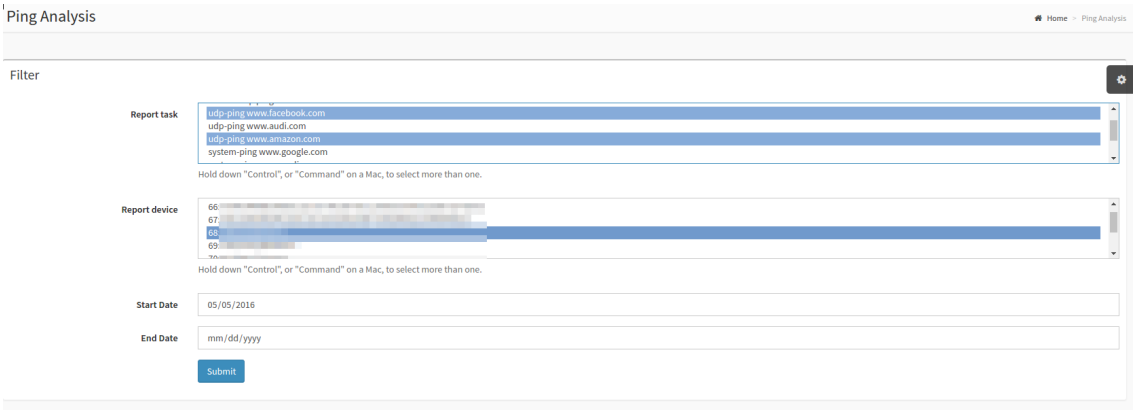


Figure 5.3: Filters for ping results

Figure 5.3 shows the filters to display results for latency measurements. Those allow to select one or multiple tasks and devices as well as the timeframe in which the results were produced.

There are two types of charts used to visualize measurement results. Figure 5.4

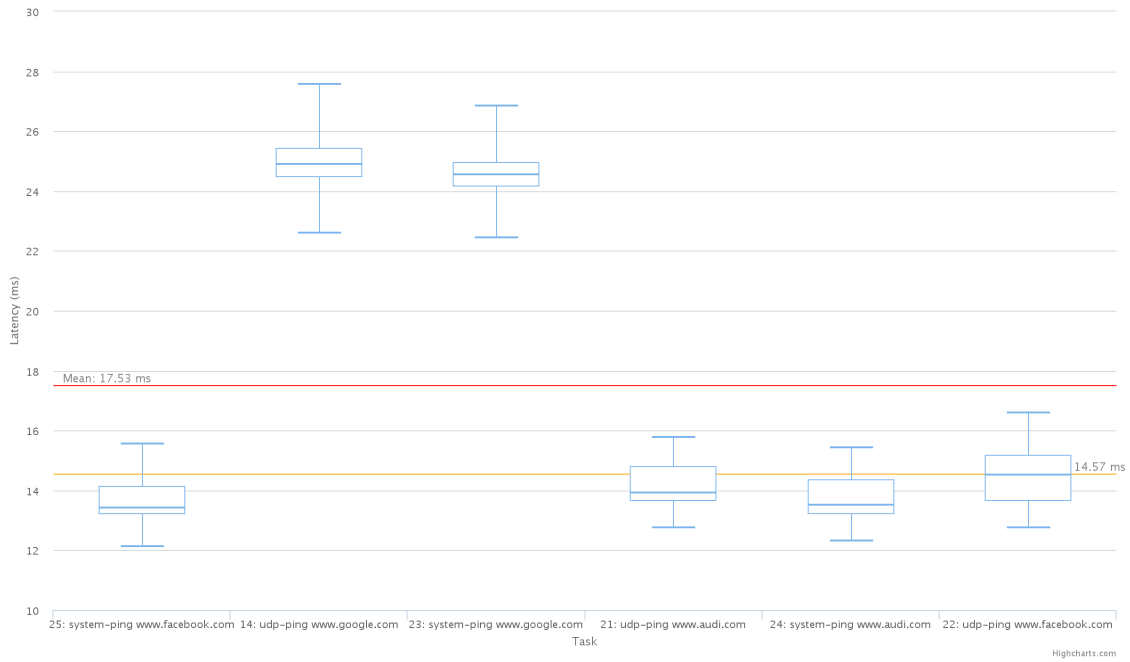


Figure 5.4: Box-plot for ping results

shows a box-plot grouped by tasks with the bottom and top of the box being the first and third quartiles, the line inside the box as the mean and the whiskers representing the highest and lowest value within 1.5 of the interquartile range. Hovering over the box gives the numbers for these metrics. The two lines and values in the chart show the calculated mean (upper line) and median (lower line) over all values used for these box-plots. Outliers are not taken into consideration for the whole chart and are subtracted out of the data before calculation.

The second chart type, seen in figure 5.5, is a scatter-plot. Each point represents the average RTT of one latency measurement, hovering with the mouse over one point brings up the numerical values.

Both charts visualize the same data: latency measurements to six different targets every hour for five days from one probe. These were done to test the measurement method and charts, but also show that (at least for this probe) the latency to the servers used for www.google.com respond much slower than the others (at least for these latency measurements). It is also visible that using the systems ping tool

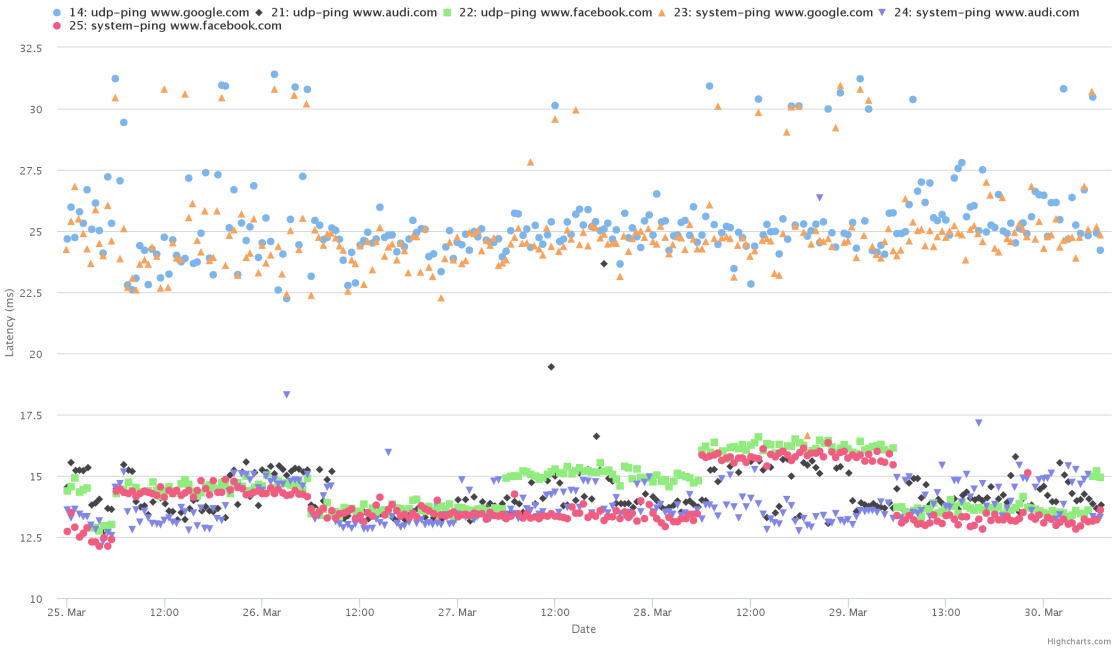


Figure 5.5: Scatter-plot for ping results

leads to a lower latency compared to when using the UDP ping implementation of GLIMPSE—even though the differences is below one millisecond on average. Details on this measurement campaign and some other performed with GLIMPSE can be found later in this chapter.

These filters and result graphs are implemented for the ping and HTTP download measurement results.

5.2.5 Website

In contrast to the dashboard for administrators and experimenters explained in the last section, the GLIMPSE website provides end-users a private area to get an overview of their deployed probes. It provides the same result charts as in the dashboard, but only for the user's own devices. Additionally, a list of all active measurement instructions for each device is visible.

To attract more users, the possibility to schedule individual measurements for a user's own devices was introduced. This allows to specify latency measurements with a periodic timing to be executed, which for example can be used to monitor the user's own infrastructure. It remains future work to add other measurement methods and timings to these user-defined measurements.

5.2.6 Probe

Measurements are executed on the GLIMPSE probe, a piece of software installed on end-hosts. This app is publicly available pre-compiled in various software repositories[2], the source-code is hosted on GitHub[3] under the BSD license.

As explained before the probe was developed with the help of Qt, which allows to write the software once and compile it for many different platforms like Windows, Linux or OSX and mobile systems like Android and iOS. Most parts are written with functions provided by the Qt libraries, only some platform dependent code was necessary, especially when handling information from the operating system like getting the CPU utilization or details from the wireless interface.

To use the software, each user has to register with the GLIMPSE back-end, which can be done directly with the app. It is also possible to do an anonymous registration where no data needs to be given, but choosing this makes it impossible to access the results on the website, add multiple devices to the account and schedule own measurements. Also, the gathered results can not be linked together after a re-installation of the application.

The probe itself pulls the configuration and measurement instructions as described before and executes them. A scheduler and multiple controllers were implemented to make sure measurements are executed and results are sent back at the specified point in time. Each probe periodically gets the offset between the local system time and the time from a remote server using the Network Time Protocol (NTP). This makes it possible to coordinate measurements across multiple probes if necessary.

It is possible to limit the traffic the probe uses for measurements. This is especially of interest for users with limited high-speed traffic volume, but also to make sure GLIMPSE does not affect other applications on the same device or in the same network by measuring all day long. Once activated the so-called traffic budget manager can limit the traffic per month in general or on a mobile interface like LTE, e.g. to 200 megabytes per month. It is also possible to disable measurements on the mobile interface completely. Once the traffic limit is reached control messages are still sent and received by the probe, but measurements are no longer executed. As shown before, every measurement result includes information from the traffic budget manager like the current values of the counters as well whether it is activated at all or not. These counters are transfered even if the traffic limitation is not activated

[2] `https://www.measure-it.net`
[3] `https://www.github.com/HSAnet/glimpse_client`

to give the experimenters an overview of the traffic they produced while measuring.

Some measurements (which are described later) use another GLIMPSE probe as peer of the measurement. For this reason a probe can be set to be passive, i.e. it does not receive measurement instructions and execute measurements itself, but only works as a counterpart for others.

As users might have different environments where they want to install GLIMPSE, two versions for separate deployment scenarios are available.

GUI Version

The graphical version of GLIMPSE was implemented to showcase the different measurement methods on mobile platforms. It runs on all operating systems but is mainly used on Android for demonstration. Besides a graphical login screen and log overview three additional functions are available.

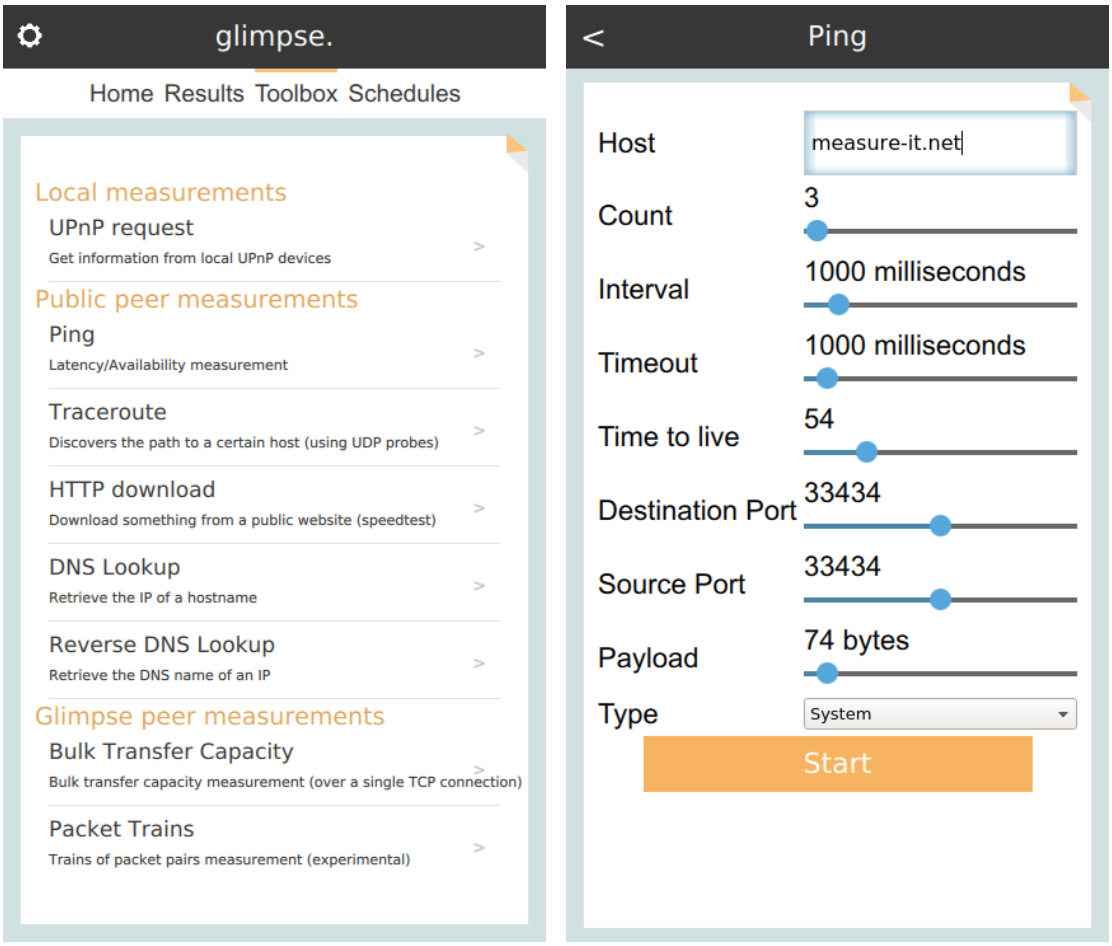
The toolbox allows users to execute measurements on-demand with individual parameters. A screenshot of the toolbox with an example of a measurement can be seen in figure 5.6.

Results from these measurements are only stored locally on the probe and are not reported back to the back-end. This might be done in later versions of GLIMPSE, but was not implemented for now because there were no resources to evaluate these individual measurements within the project. Also, not reporting these allows users to perform "private" measurements using GLIMPSE.

The measurement history shows a tabular view of all executed measurements on the probe as seen in figure 5.7a. This is used to provide a quick overview of executed measurements and for debugging purposes. A graphical visualization of the results is done in the user area of the GLIMPSE website. Later versions of GLIMPSE might also include visualizations within the app itself.

Figure 5.7b shows the settings view of the app. The aforementioned traffic budget manager as well as the device name, backlog and whether or not to use Google Analytics can be configured there. Google Analytics is used to anonymously track the distribution of the probe as well as to get information on how users interact with the app (e.g. which screens are used most).

Finally, the scheduler tab allows to see all scheduled measurements for this probe. This includes the ones specified by the administrators of the platform as well as the ones the user has defined for this probe in the user area of the GLIMPSE website.



(a) Overview **(b)** Ping

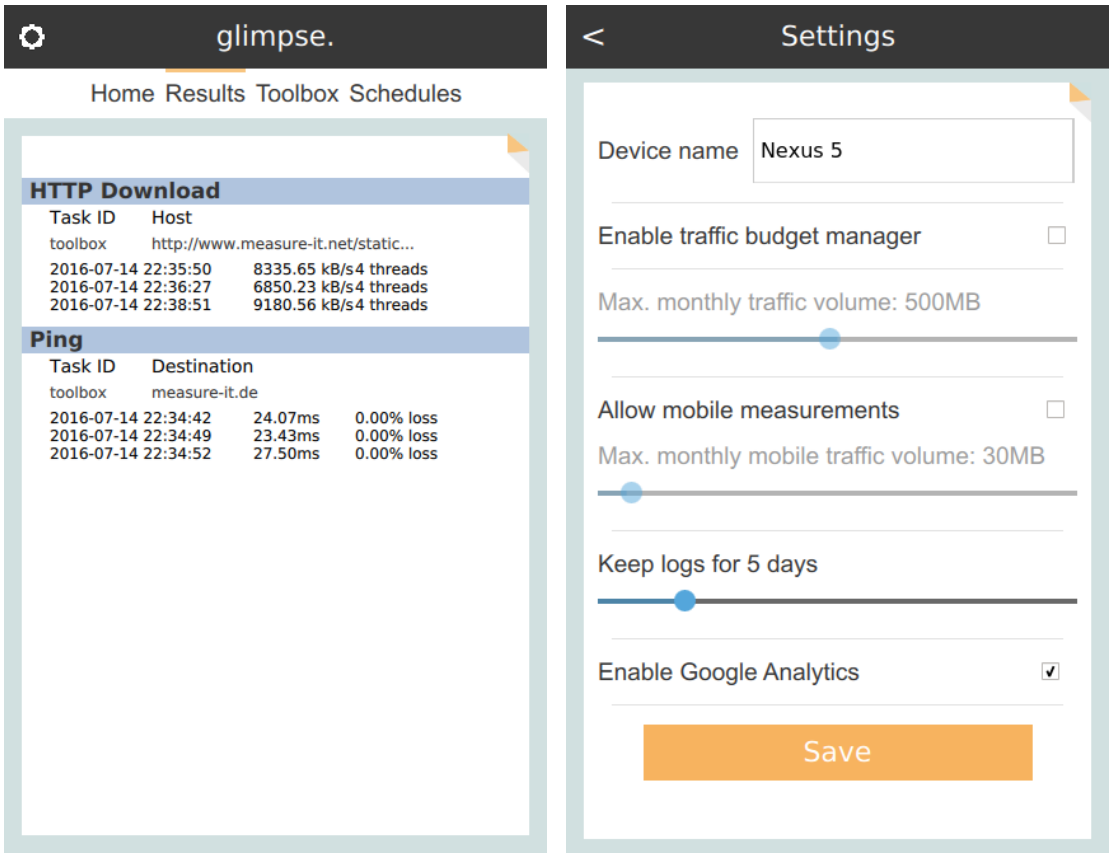
Figure 5.6: GLIMPSE toolbox

Console Version

The console version is a headless version of the GLIMPSE probe without a graphical user interface. It is build to be used on servers or single-board computers where the app can run continuously in the background as a system service. The project uses BeagleBone Blacks[4] and Raspberry Pis[5] to distribute pre-installed probes for the homes of interested participants. They have the advantage of being low-cost and small enough to deploy right next to the home gateway.

This version does not have the possibility to execute measurements on-demand like the GUI version does. Measurement results can either be viewed on the website

[4] `https://beagleboard.org/black`
[5] `https://www.raspberrypi.org/`



(a) Results **(b)** Settings

Figure 5.7: GLIMPSE App

or the admin dashboard, or the raw JSON result files in the local file system.

There are some command line parameters available to set the probe up, there is no other interaction possible.

5.3 Measurement Methods

The measurement methods provided by GLIMPSE are described in this section. Some of them are still considered experimental and only used in lab environments. Nevertheless, all of them are implemented in the publicly available application and can be used for measurement campaigns.

Every measurement takes a list of parameters as input, which affect how the measurement works. The result fields for each measurement are predefined. Although not every field needs to be filled-in by the measurement, no additional fields can

be added. This is necessary when using a relational database as sink for the result data, as the tables for storage and their relations need to be created in advance. Furthermore, this makes comparing the results from multiple measurements by the same method easier.

Each execution of a measurement runs in its own context, results from previously executed measurements do not influence the outcome. This is of course only true from an internal point of view. E.g. if a measurement triggers some rate limiters subsequent measurements are in fact affected. But it is not possible to access data from previous measurements within a measurement method.

To encapsulate measurement methods from the rest of the code of the application, a plug-in-like system was designed. This would also allow external developers to write measurement methods for GLIMPSE without knowing all technical details of the app (which remains future work). The probe also ensures that only one measurement can be executed at the same time. If the starting time of multiple measurements overlap the probe executes them one after another and stores the starting times with the result (as seen before). When undertaking time critical measurements (e.g. coordinated load testing from multiple probes) the experimenter has to review the starting times in each measurement result.

5.3.1 Latency

In general, a latency or ping measurement tries to determine the round-trip time (RTT) between two hosts in a network. For GLIMPSE, the latency measurement has multiple parameters to influence various parts of the measurement.

The *host* parameter sets the destination of the ping, which can be an IPv4 address as well as a URL. *count* specifies the number of ping packets send as part of this measurement, *interval* and *receiveTimeout* determine how long to wait between two sent packets and how long to wait for an answer to each packet. GLIMPSE provides three latency measurements using different types of packets, differentiated by the *type* parameter. Their characteristics and implementations are described in the following paragraphs.

The classic ping every Internet host must implement as defined by RFC 1122 uses ICMP Echo Request packets. A host should reply with an ICMP Echo Reply packet once a request was received. However, opening raw ICMP sockets which are necessary for sending and receiving ICMP Echo packets is normally not possible without privileges on Linux. There are some workarounds for this, for example using

a special kind of ICMP socket which is only possible if some system variables are set[6]. But as described before, one of the design goals of GLIMPSE is that it should run without root privileges on all platforms. Fortunately, the Linux ping-tool can be executed by every user, which is also true for all other target platforms of GLIMPSE. Therefore, the first *type* of the latency measurement in GLIMPSE is **system**, which executes the system's ping tool and parses the output into the GLIMPSE result format (as described later).

To have more control over the measurement and to be able to determine a round trip time even if the source or destination does not support ICMP packets, two other types were implement, one using TCP, the other UDP packets. Both add four additional (and optional) parameters to the measurement: the *sourcePort* and *destinationPort* to set the ports to be used, *ttl* to set the TTL value of the packets and finally *payload* to influence the payload length (padded with random values). Furthermore, both allow to use IPv6 addresses as the destination.

The **TCP** type measurement is implemented by just trying to connect to the given destination and port by sending a TCP SYN packet to the destination. If an answer is received (which might be a SYN ACK or a RST packet) the time from sending to receiving is calculated and used as round-trip time just as the ICMP Echo Request and Reply would do. Not every host generates answers to a SYN packet when no actual service is running on the machine with the given port. Therefore, if no reply is received within the set receive timeout, the measurement has failed and no round-trip time is returned. This measurement has the advantage of working with every destination known to have a publicly available TCP service running (which is true for every web server). However, the research by Weichao, Li et al. showed that using TCP SYN packets (and also ICMP packets) to measure latency on Android devices can lead to increased round-trip times due to the physical link or the driver reacting slower to them [LMWC15]. It remains future work to implement a latency measurement using TCP data packets within GLIMPSE and compare, if other platforms also introduce these kind of delay.

The last type of latency measurement within GLIMPSE is **UDP**. It sends out a UDP packet which is either answered by a UDP packet in case of a service running at the given target port and destination, or by an ICMP packet indicating the error which occurred. Those answers can be used to calculate the round-trip time. It is also possible that the packets are blocked or that no answer is generated at all in which

[6] `http://stackoverflow.com/a/20105379`

case a receive timeout occurs and no round-trip time can be determined. To receive an ICMP packet for the sent UDP packet the error message queue is used in Linux-based operating systems and libpcap for Windows, in OSX an ICMP socket can be opened directly. The type of the received ICMP packet helps to distinguish if the error was on the path (for example by an ICMP Time Exceeded or Destination Host Unreachable error), in which case no round-trip time to the target can be calculated, or if the actual destination answered (Destination Port Unreachable error).

All of the three latency measurements produce the same result values. Firstly, the measured round-trip time in milliseconds for each packet is provided in a list. To take load of the back-end and save time while analyzing the data, some aggregations for this list are pre-calculated directly on the probe and also stored in the result. These include the average, minimum, maximum and standard deviation. The number of packets sent and received as well as the loss rate are also included. While using the latency measurement with destinations set to a URL, it became apparent that the IP of the destination should also be included in the results. For example when pinging *www.google.com* the used server might be different depending on the DNS server and geolocation. To make results to the same destination comparable, the actual IP used for the ping when a URL was given is also included in the result.

Listing 5.4 shows an example of a latency measurement result. It includes the actual destination IP used to ping (which is useful when using a DNS name as target), all round-trip times measured and multiple pre-calculated statistics like the average and standard deviation. To determine the loss rate, the numbers of packets sent and received are also saved.

5.3.2 Traceroute

The traceroute measurement of GLIMPSE utilizes the previously described UDP latency measurement by sending out multiple packets and reacting to the response (in form of either an ICMP or UDP packet) accordingly. Every time an ICMP TTL Exceeded error is received, the TTL is increased by one (until the hard-coded value of 20 to prevent the measurement of taking to long). If an ICMP Destination Unreachable error or a UDP packet is received, the end of the route is found and the measurement ends. The same parameters as for the latency measurement except the TTL and type parameter can be set for the traceroute: destination host, destination and source port, number of packets sent for each hop, time interval between each packet sent, receive timeout and payload length. For every hop, the round-trip times

```
"destination_ip": "74.125.206.105",
"round_trip_ms": [
  18.099000930786133,
  26.618999481201172,
  22.343000411987305,
  17.891000747680664
],
"round_trip_loss": 0,
"round_trip_received": 4,
"round_trip_avg": 21.238000869750977,
"round_trip_sent": 4,
"round_trip_stdev": 3.5788200403009056,
"round_trip_min": 17.891000747680664,
"round_trip_max": 26.618999481201172
```


Listing 5.4: GLIMPSE latency measurement result

and the response types (for example TTL exceeded or destination unreachable) as well as the same aggregate values as for the latency measurement (like the average and standard deviation) are stored in the result.

5.3.3 HTTP Download Speed

To measure the available bandwidth, the HTTP download speed measurement was implemented. It uses HTTP for two reasons: (a) many speedtests available on the Internet utilize HTTP for their measurements and GLIMPSE wants to be comparable to them and (b) when users measure their bandwidth the results should reflect what they can expect when typically using the Internet, which is browsing and downloading using HTTP. Implementing a non-standard application layer download measurement, for example a byte-stream directly over TCP, is not an option when taking these aspects into account.

However, GLIMPSE also implements bandwidth measurements directly with TCP and UDP, which have some advantages over the HTTP measurement method at the cost of being less flexible when choosing a measurement target. The details of this method are described in the next section.

The options available for this measurement method are:

URL The HTTP download speed measurement downloads a file from a publicly available HTTP server specified by this URL. Note that the file needs to be

sufficiently large to not be completely downloaded in the target time (see below) specified for this measurement.

Source port The destination port of the web server can be set within the URL given, the source port used to download is determined by this parameter (or obtained randomly if not set).

Thread count One or multiple threads are used to download the file in parallel. As previously done research has shown it is necessary to use multiple connections to obtain reliable values for the line capacity [RTMM14].

Ramp-up time To prevent measuring while TCP is still in the slow start phase of the congestion control algorithm [APB09] a ramp-up time can be specified after which the actual measurement should start.

Target time The measurement tries to download the file for a complete duration of the ramp-up time plus the time set with this parameter. If the target time can not be achieved (for example when the file to download is not large enough) this measurement is marked as tainted as described below.

Avoid caches To prevent caches from serving the content of the requested file a timestamp is added to the request when this parameter is set.

Slot length Multiple samples of the download speed are taken per second. To reduce the size of the result only the means of all samples within each slot length (in milliseconds) are stored.

Listing 5.5 shows a shortened example of an HTTP download speed measurement result in the internal GLIMPSE result format (which is JSON). It includes multiple things:

actual_num_threads The actual number of threads used to download, as this number might differ from the thread count requested for the measurement if the operating system has not enough resources to create all necessary threads.

destination_ip The IP of the server used to download the file.

bandwidth_bps_avg The summed up mean bandwidth over all threads in bytes per second.

```
"actual_num_threads": 4,
"destination_ip": "212.114.242.164",
"bandwidth_bps_avg": 87754555.22097653,
"bandwidth_bps_per_thread": [
    {
        "max": 34001677.63862328,
        "min": 15904394.591750149,
        "avg": 27621239.50298537,
        "stdev": 5037227.330890204,
        "slots": [
            15904394.591750149,
            32680961.69489387,
            34001677.63862328,
            32281004.47267595,
            17128994.99331912,
            22626819.001601294,
            24613372.12222783,
            29924572.736371152,
            29234613.335270293,
            28172257.874825772,
            32107857.438503873,
            33299442.16343996,
            25180010.087120537
        ]
    },
    {
        "max": 23435616.60051421,
        "min": 16807007.576765336,
        "avg": 18791416.77791934,
        "stdev": 5914968.405935366,
        "slots": [
            [...]
        ],
    },
    {
        [...]
    },
    {
        [...]
    }
],
"results_ok": true
```

Listing 5.5: GLIMPSE HTTP download speed measurement result

bandwidth_bps_per_thread The average and some other aggregations for the bandwidth for each thread in bytes per second.

slots in bandwidth_bps_per_thread A list of the means for each slot in each thread in bytes per second.

results_ok A boolean flag to indicate if the results are trustworthy. This flag is only true if all started threads (note: not requested threads) could measure for a period greater than 75% of the requested target time.

The following parameters were used for this particular measurement result: source port 33434, thread count four, ramp-up time two seconds, target time five seconds, avoid caches true and slot length 500 milliseconds.

5.3.4 Bulk Transfer Capacity (BTC)

This measurement method utilizes TCP packets to determine the bulk transfer capacity to a destination without the overhead of an application layer protocol header like HTTP using a single TCP connection [PDMC03]. Unfortunately, this comes at the cost of not being able to use any publicly available file from an HTTP server as target to download which is possible for the HTTP download measurement described before. It is necessary to have a so-called measurement peer (MP) understanding this GLIMPSE measurement method on the other side which acts as a server. An MP opens a TCP server socket and waits for new connections from a GLIMPSE probe, which requests in the initial packet the number of bytes it wants the MP to send back for this measurement. The actual measurement takes place in two steps: First, the probe requests a small amount of data to be sent by the peer, the `initial_data_size`. The measured speed from this transfer is used to determine the real available bandwidth by requesting enough data from the peer to download for three seconds with the predetermined speed. This is in contrast to the HTTP download measurement method, where the options allow to specify the time for how long to download from the server, meaning the file used to download has to be large enough to not transfer it completely within this time. The bulk transfer capacity (BTC) measurement uses the initial data size to dynamically determine the needed amount of data to download which is simply generated. But this also means this option has to be chosen carefully to produce meaningful results for the actual connection speed. Choosing a too small initial data size will lead to incorrect results,

while a too large initial data size will stall the probe with this measurement for a long time (for example when requesting to download 10 megabytes on a 64 kbps connection). Therefore, this measurement should only be used when the bandwidth of the connection can be roughly estimated in advance.

The idea when developing this measurement was to use other GLIMPSE probes as peers for this measurement. This is not practical in the general case when assuming that most probes are in the premises of end-users, as the available upload speeds are much smaller than the typical download speed of a residential Internet access line. Also, this usually implies connecting to a probe behind a home gateway which typically implement NAT and it is not possible to connect to the probe without doing some kind of NAT traversal [GF05].

The available parameters for this measurement are the *host* and *port* to use as measurement peer, the *initial data size* as described before and the number of *slices* to use to group the result values in.

The measurement results include the determined download speed as well as some pre-calculated aggregations as seen before.

5.3.5 DNS and Reverse DNS Lookup

DNS and reverese DNS lookup are two rather simple measurements using a similar technique. The DNS lookup takes a URL or host name and returns the DNS entries available, which typically leads to the IP address of the given target. In fact, the implementation attempts an ANY (QTYPE) DNS request retrieving all records for the given name. The nameserver to use can be set, otherwise the default local resolver is used for the lookup. An example result can be seen in listing 5.6.

The reverse DNS lookup measurement does (obviously) the reverse: it takes an IPv4 or IPv6 address and tries to determine the domain name for this IP. A lookup for 216.58.208.46 would for example lead to “fra15s12-in-f14.1e100.net”, the host name of a machine from Google probably located in Frankfurt (as indicated by the airport code FRA at the beginning of the host name).

5.3.6 Universal Plug and Play (UPnP) Discovery

The UPnP protocol has the ability to retrieve information about devices in the local network. This protocol is used by many home gateways and media devices like TVs or Network Attached Storage (NAS) devices to e.g. offer information about

```
"records": [
    {
        "name": "google.com",
        "ttl": 299,
        "type": "A",
        "value": "216.58.208.46"
    },
    {
        "name": "google.com",
        "ttl": 299,
        "type": "AAAA",
        "value": "2a00:1450:4001:815::200e"
    }
]
```

Listing 5.6: GLIMPSE DNS lookup result

the services they provide. The measurement implement in GLIMPSE is restricted to gather information only from home gateways providing the Internet access for the local network. Not every home gateway has UPnP enabled by default or even supports it at all, but it can provide many useful information as seen in the following list:

Connection information External IP address, connection type, maximum bit rate of the link layer

Counters Bytes sent, bytes received, packets sent, packets received

Device information Model name, manufacturer, device name, uptime

Miscellaneous Port mappings, firewall status, firewall pinholes

Especially the byte counters are useful to identify cross traffic while measuring by reading these values before and after a measurement. Also the link layer rate is interesting when evaluating bandwidth measurements.

5.4 Future Work

Even after the three-year project lifetime, GLIMPSE is still being extended by the Networking Group of the University of Applied Sciences Augsburg. There are a

number of features which would improve GLIMPSE and would enable a greater variety of measurement campaigns.

At the moment, the measurement methods used are not actually loaded dynamically as plug-ins, but are hard-coded into the application. The code is already prepared to make a change to support dynamically loaded measurement methods as every method is already encapsulated and sufficiently isolated from the rest of the application and could run independently from GLIMPSE. It remains future work to actually implement a plug-in system, but also to write a documentation for developers and researchers on how to contribute self-written measurement methods to GLIMPSE without in-depth knowledge of the platforms internals. Additionally, a public interface could be provided to submit and review those new methods. The app itself could also be extended in a way so that a user can list, download and select the measurement methods the user wants to allow to be executed on a device. This would take GLIMPSE a step further to a multi-purpose research platform in which researchers could inject and advertise their own, individual measurement campaigns. The general spirit of this would be along the lines of the BOINC platform [Uni], which has used this approach quite successfully for volunteer computing instead of network measurements.

To make GLIMPSE more powerful for the end-users, future work could also include more methods to be allowed for user-defined measurements. In order to test this feature, currently only the latency measurement is available for users to schedule for their own devices at the moment. Other measurement methods could be enabled easily, but for lack of time this was not implemented yet. Furthermore, the temporal scopes to use and the selection of participating devices could be improved to allow end-users to have complex schedules to execute measurements on a subset of their devices, or even to allow execution only if certain conditions are met (e.g. only execute the measurement on devices running Microsoft Windows). This could be done with the already implemented preconditions and qualifications.

Certainly, the graphical user interface of GLIMPSE could be improved. The focus of development was on developing the core system and testing the scheduling and the execution of the measurement methods including the communication with the back-end components. To release the app in the Google Play Store or the Apple Store more polishing needs to be done, which would require some graphic designers and front-end developers.

Finally, more research and development with the reasoner-system is possible. This

feature was added very late to the project and has been implemented only with an example reasoner to test and understand the needs for this system. Therefore, analyzing the results of the already written traceroute reasoner and adding additional reasoners remain future work.

6 Measurement Campaigns

This chapter describes and discusses the measurement campaigns conducted as part of this dissertation in 2015. Most of the presented work here has been done solely using the GLIMPSE measurement platform. The first section 6.1 however describes an experiment where GLIMPSE was used to conduct control experiments as part of a study to measure the accuracy of the Speedtest.net service.

The box-plots shown in this chapter are taken directly from the GLIMPSE dashboard and show the same information as described in section 5.2.4. For convenience, this information is repeated here: The bottom and top of the box show the first and third quartiles, the line inside the box is the mean and the whiskers represent the highest and lowest value within 1.5 of the interquartile range. The two lines and values in the middle of each chart show the calculated mean and median over all values used for the box-plots. Outliers are not taken into consideration for each chart and are removed from the data before calculation.

The scatter-plots shown here are also taken from the GLIMPSE dashboard, where each point represents the average speed for one HTTP download measurements or the RTT for one latency measurement.

6.1 Speedtest.net

Speedtest.net, a service operated by Ookla is essentially a website provided to end-users to measure their current bandwidth to and from a close-by server located somewhere on the Internet [Ookb]. This measurement is implemented as a Flash plugin for the web browser. Having it available through a browser is convenient as browser-based tools are cross-platform by default and reach a broad user-base instantly. The flip side of browser-based tools is the potential inaccuracy, as they are sand-boxed and run inside of the browser, which adds overhead. In order to determine how accurate this particular browser-based measurement is, an experiment at two different residential broadband networks was conducted. Both had an Internet

connection with 100 MBit/s download and 10 MBit/s upload bandwidth and were located in Augsburg, Germany through different ISPs. At both locations, a computer was set up connected via Ethernet directly to the home gateway in order to avoid effects introduced through e.g. WiFi. Note that cross traffic could be present in all measurement results shown here, as the network was used by the users of the network as usual (which should also be expected when using Speedtest.net in a network with other users).

Before a Speedtest.net measurement can begin a server has to be selected against which the measurement takes place. This selection process is based on location (selection of the candidate set) but ultimately on latency to the measurement servers as a measure for topological distance. This is an automated process, but manual selection of a server is possible.

As Speedtest.net is a Flash plugin running in the browser, the automation of starting tests is not easily possible. The website has to be loaded and the mouse has to be used to optionally select a test server, begin the test and screen-scrape the results. To not have to do this manually a script for Sikuli [Sik] was written, which automates this process. This script opens the Google Chrome webbrowser with an incognito tab for `http://www.speedtest.net` to make sure to have a fresh browser context without previously set cookies. Sikuli performs image recognition to identify and control GUI elements, in this case for the the "BEGIN TEST" button of the Flash plugin, clicks on it and waits until it recognizes that the screen is showing the result screen. In case that this does not happen for 120 seconds the measurement is aborted. Afterwards, an image of the result view is stored and Python-tesseract [Mat] is used to get the measured speed values with optical character recognition. This is necessary because Speedtest.net does not provide a text output for the results but only non-copyable images. These results were written in a CSV-file and later compared with the images to make sure the recognition did work. Unfortunately, automatic optical character recognition for these results did not work very well, therefore every result had to be verified by hand. For this reason it was decided to conduct this experiment every hour for only 48 hours in location A and focus on the other location (B) for a time of two weeks.

The chart in figure 6.1 shows the results for location A. The average download speed measured was at 80.53 MBit/s with a maximum (minimum) of 93.75 (32.10) MBit/s. Interestingly, the average (12.12 MBit/s) and maximum (22.05 MBit/s) upload speeds measured were above the contractually available bandwidth for the

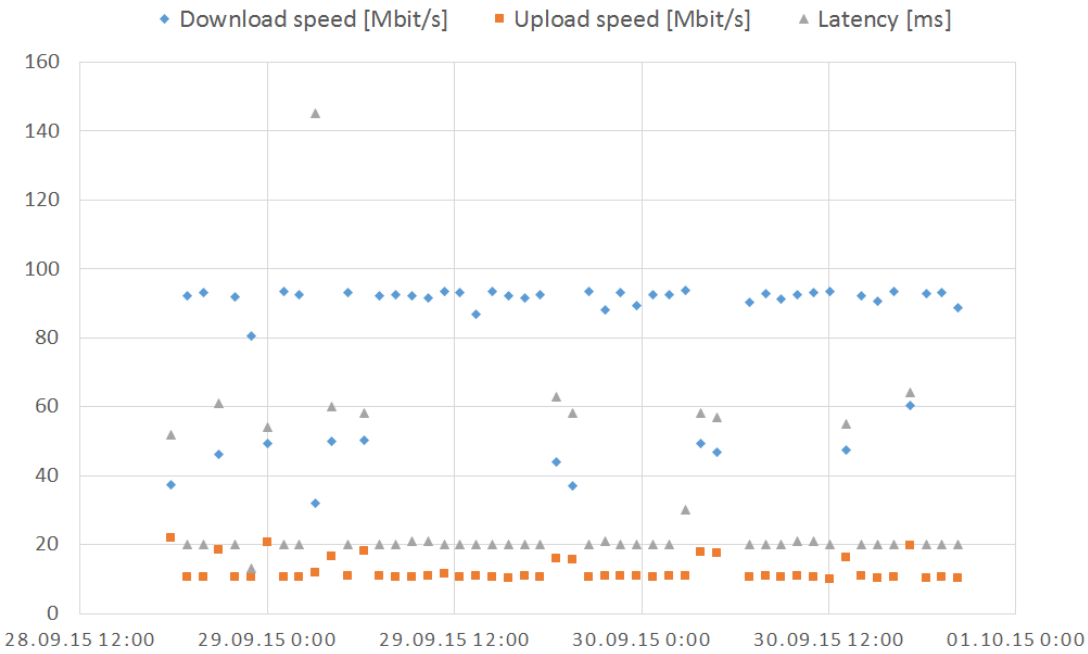


Figure 6.1: Result chart for location A

connection (10 MBit/s). When looking at the chart, the upload speeds are only above average when the download speed in the same measurement was way below the average. Also, in all of those cases, the latency, which had an average of 31 milliseconds, doubled. After manually looking at the stored result images another observation was made. For most of the measurements a server in Amsterdam, Netherlands was chosen to obtain the values. All but one measurements with high latencies, low download and (too) high upload speeds were made against a different server located in Harderwijk, Netherlands. It is not known why servers in the Netherlands where chosen, Speedtest.net [Ooka] describes the process of selecting the server is based on the determined location of the client. After selecting the five nearest servers, the latency to each one is measured and the server with the lowest result is selected. Only one measurement with similar bad performance was observed which did not use Harderwijk measured against a server in Pfaffenhofen an der Ilm, Germany, which is within 100 kilometers of Augsburg. Both locations in the Netherlands are over 700 kilometers away. Table 6.1 gives an overview of the described average results by server sorted by number of measurements against the respective server.

For a large fraction of the measurements at location A, the server selection algorithm of Speedtest.net picked a suboptimal server which consistently resulted in significantly lower downlink bandwidths displayed to the user. A quarter of the mea-

Location	Download [MBit/s]	Upload [MBit/s]	Ping [ms]	Count
Amsterdam	92.10	10.65	20	36
Harderwijk	46.62	17.13	67	11
Ingolstadt	80.42	10.69	13	1
Pfaffenhofen / Ilm	37.30	22.05	52	1
Average / Sum	80.53	12.34	31	49

Table 6.1: Result table for location A

surements would have indicated less than half of the contracted downlink bandwidth, which certainly would be a reason for complaint by the user. Additionally, it cannot be said with certainty if the upload speeds above the contractually available value are measurement errors or a temporary misconfiguration by the ISP. These values could not be reproduced when trying to upload a file manually to a known server and were not observed by the user of this Internet connection before.

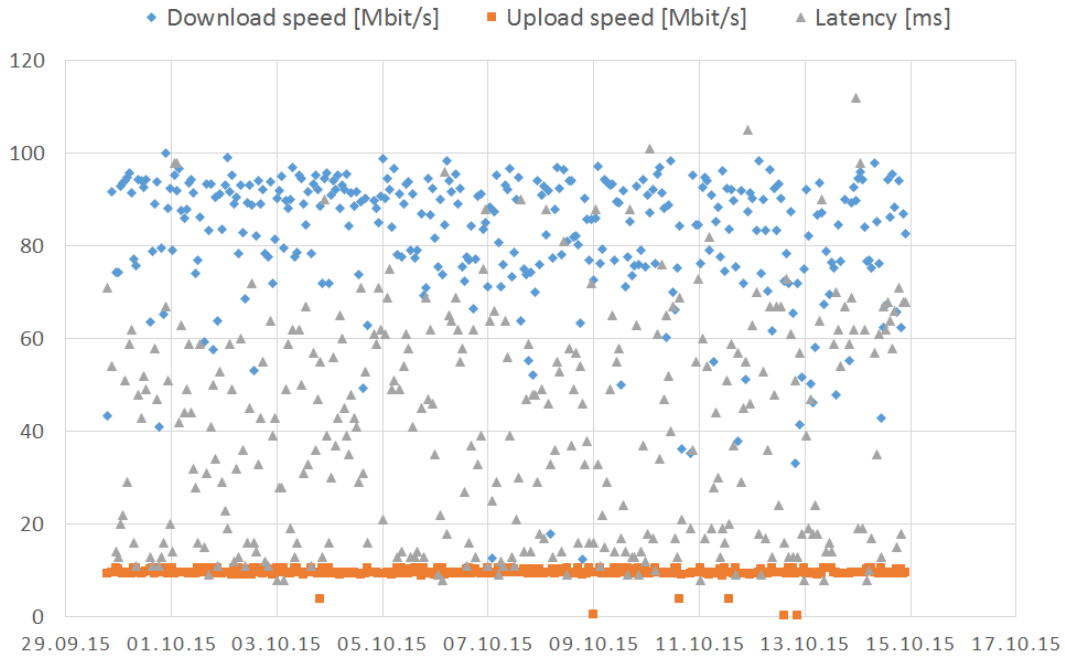


Figure 6.2: Result chart for location B

Figure 6.2 shows the result chart for location B during a measurement period of two weeks. The latency results are not as consistent as for location A, but there are no spikes for the upload speed at this location. The average download (82.20 MBit/s)

and upload (9.58 MBit/s) bandwidths were similar however, but the average latency with 40 milliseconds was 10 milliseconds higher. There seems to be no correlation between latency and measured speeds by just looking at the chart. When grouping the results by servers used it can be seen that all servers are located in Nurmberg, Germany, which is about 170 kilometers away from Augsburg. Table 6.2 shows the average results for each server, this time not named by location but by the company which provided the server.

Location (Nurmberg)	Download [MBit/s]	Upload [MBit/s]	Ping [ms]	Count
Expert-Webservice	90.62	9.45	49	127
Linux.GQ	76.85	9.45	48	80
Pkern.at	84.81	9.10	56	44
QuESD	70.30	9.66	15	38
123 Commerce Ltd.	75.59	10.43	13	34
ratiokontakt	77.45	10.45	9	17
Average / Sum	82.20	9.58	40	340

Table 6.2: Result table for location B

The server used most of the time lead to the best download measurement results and influenced the overall average bandwidth positively. Interestingly, and in contrast to location A, the latency against this server was the second worst on average. As Speedtest.net selects the server out of five close-by servers by choosing the one with the lowest latency it is obvious why this server was not used for the bandwidth measurement every time. Also notable is the fact that the result table shows six servers used for measuring. This could be due to various reasons. E.g. a new server was added during the measurement campaign, the logic to determine the location of the client re-classified the location of the client, or the logic selecting the candidate server set does not always result in a deterministic set of servers, in particular if there are multiple servers in the same location as was the case here. Interestingly, the servers with the best latency results (9, 13 and 15 milliseconds) lead to relatively bad download speeds (70.30, 75.59 and 77.45 MBit/s) and good upload speed (9.66, 10.43 and 10.45 MBit/s).

Multiple GLIMPSE HTTP download measurements to different servers under the

experimenters control were executed at location B in the same measurement period to be able to compare the results against the ones obtained from Speedtest.net. In other words, GLIMPSE probes performed control experiments to put the obtained results into context. Figure 6.3 shows the scatter-plot for download speeds from a server deployed in the data center of the used ISP. This means this server is geographically and (network-)topologically near location B (100 kilometers respectively five hops).

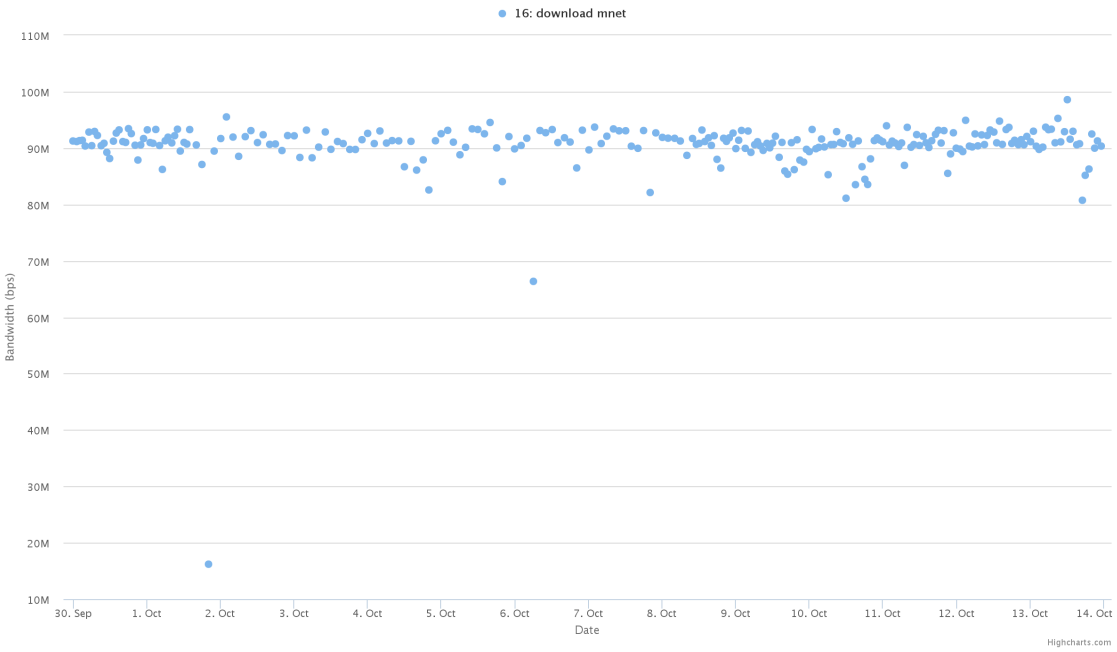


Figure 6.3: Scatter-plot for the results of a HTTP download measurement in location B

The average measured speed was 91.09 MBit/s, it is clearly visible that the variation of the results is much smaller. In principle, this could be due to many factors. E.g. the measurement implemented in GLIMPSE has access to the socket interface and high precision timers directly and therefore, it can perform measurements with an accuracy that cannot be obtained in a Flash plugin. Also, there is no variability introduced through, e.g. the browser and the Flash engine. Based on the fact however that bandwidth estimations decisively differ by location, as an overall conclusion for this measurement campaign it can be said that the quality of the results for Speedtest.net greatly depends on the selection of a server to measure against. Using servers with low latency is not always the best server to use for measuring download or upload speeds and a better selection algorithm should be devised since the automatic process of selecting a server by Speedtest.net seems to return bad results

for a non-negligible fraction of the measurements. The current rate of poorly picked servers is worrisome because for bandwidth estimations, Speedtest.net is somewhat of an authority.

6.2 HTTP Download

All measurement campaigns presented in this section where conducted with the previously described HTTP Download measurement of the GLIMPSE probe. Multiple locations in homes of end-users as well as data centers were used for the measurements. Each probe was installed on identical dedicated single-board-computer devices (BeagleBone Black) with the same software versions. This includes the same operating system, kernel, and networking parameters (like the TCP congestion control algorithm and TCP parameters). All probes where connected directly via Ethernet to the (home) gateway to avoid interference introduced by a wireless interface. The Internet access rates and used access technologies (like cable or fiber to the home) differed.

6.2.1 Parameter Comparison

As described before in section 5.3.3 there are a number of parameters available for the HTTP download measurement to influence the destination and quality of the measurement:

- **URL** The location of the file to download. This parameter should include the protocol (HTTP or HTTPS) to use as well as the port.
- **Source port** If unset a random source port us used for the measurement, otherwise the HTTP GET request is executed from the specified port.
- **Thread count** The number of threads to be used in parallel while downloading.
- **Ramp-up time** The time in milliseconds the file should be downloaded before the actual measurement is done. This is used to allow the TCP slow-start to increase the window size to reach the real download speed and not include this "ramp up" into the actual measurement.

- **Target time** The actual time the download measurement should take after the ramp up time.
- **Avoid caches** To avoid caches this parameter can be set to include a timestamp into the GET request.
- **Slot length** The probes calculate the download speed multiple times per second. This parameter allows to.

The following three will influence the measurement accuracy directly: the thread count, the ramp-up time and the target time. A bandwidth estimation that works on the basis of saturating the bottleneck link between two hosts is clearly an intrusive process. Therefore, it should run as long as needed to result in an accurate estimation but not longer, to not interfere with other use of the network. After some initial tests, the default parameters were set to the following values:

- **Target time** set to 5 seconds.
- **Ramp-up time** set to 2 seconds.
- **Thread count** set to 4 threads.

These parameters lead to accurate results for probes where the available bandwidth was known and could therefore be verified. To evaluate how changing these values influence the measurement results, a measurement campaign was scheduled on all available probes. Obviously, measurements of this kind cause a substantial amount of traffic. Since probes should not be disruptive to the normal operation of the network and not overburden connections with a volume cap, parameters should be chosen that cause as little traffic as possible while still getting accurate results for the maximum bandwidth of a probe. With this in mind the following parameters where chosen:

- **Target time** was set to 3, 5 and 8 seconds
- **Ramp-up time** was set to 1, 2 and 4 seconds
- **Thread count** was set to 1, 2 and 4 threads

Only one parameter was changed at a time to isolate the effect of that parameter change.

Target Time

The box-plot in figure 6.4 shows the comparison of the bandwidth results of the three target time measurements for all participating probes. It is easily visible that the

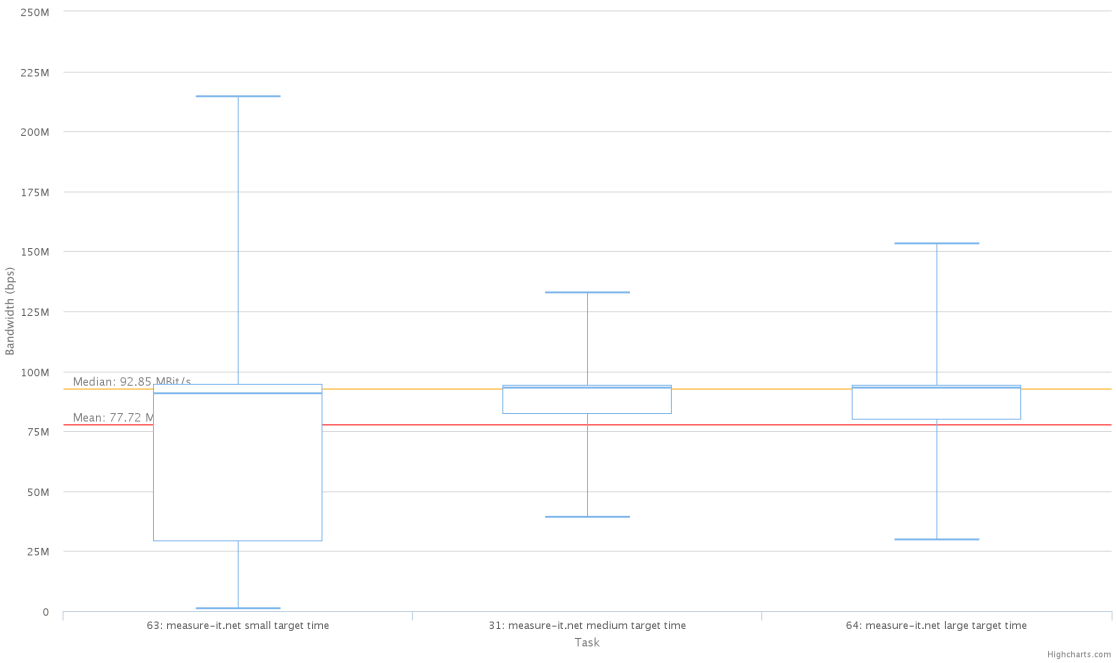


Figure 6.4: Box-plot for the results of the target time parameter comparison

shortest value for the target time (three seconds) is not long enough to obtain stable measurement results. The variation of the determined speed is very high. Note that the results for all 13 devices executing these measurements are used in this boxplot, which includes one device with a download bandwidth of about 275 MBit/s and multiple devices with 100, 50, 20 and below Mbit/s. The scatter-plot of the same data in figure 6.5 gives an idea of the different download speeds present. The graph nicely shows the different maximum download speeds, which are amongst the most commonly offered rates available at most ISPs in Germany. These results also show that the default value of five seconds for the target time is enough to get reliable results, downloading three seconds longer seems not to improve the quality of the measurement but causes significantly more traffic.

Ramp-up Time

Unfortunately, the results for the ramp-up time comparison were mostly inconsistent. Figure 6.6 shows the boxplot for a probe known to have 100 MBit/s. While the

small and medium ramp-up times show good results, the large ramp-up time shows significantly lower download rates. Normally, a higher ramp-up time should improve the quality of the results, as the bandwidth probing (slow start) part of the congestion control algorithm of TCP takes place in the first seconds of a new connection and should then not fall into the actual measurement time.

For probes with a slower Internet connection the ramp-up time parameter did not influence the measurement results significantly as seen in figure 6.7 for a probe with a known bandwidth of 10 MBit/s.

When looking at the overall box-plot in figure 6.8 the variation seems to be the lowest for the large ramp-up time, while the highest average download speed was measured when using the medium ramp-up time.

Due to these inconsistent results the default value for the ramp-up time was not changed and revisiting this measurement campaign remains future work.

Thread Count

For the last parameter, figure 6.9 shows the box-plot of the results for one device with a known download bandwidth of 100 MBit/s. This chart is exemplary for the results of the other devices and indicates that one and two threads respectively TCP

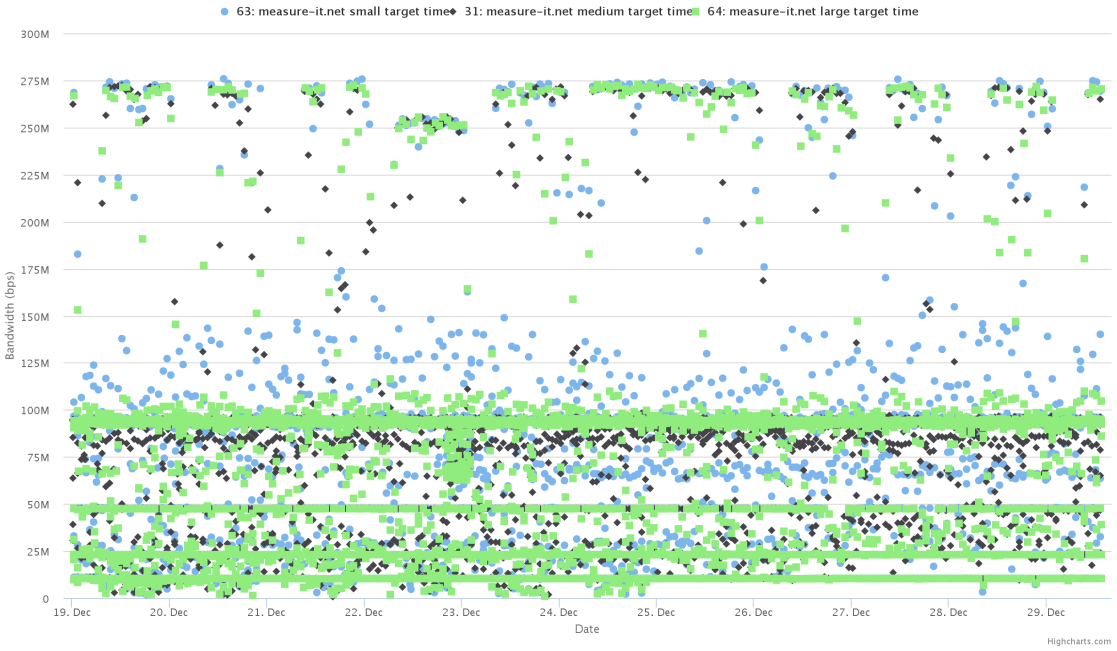


Figure 6.5: Scatter-plot for the results of the target time parameter comparison

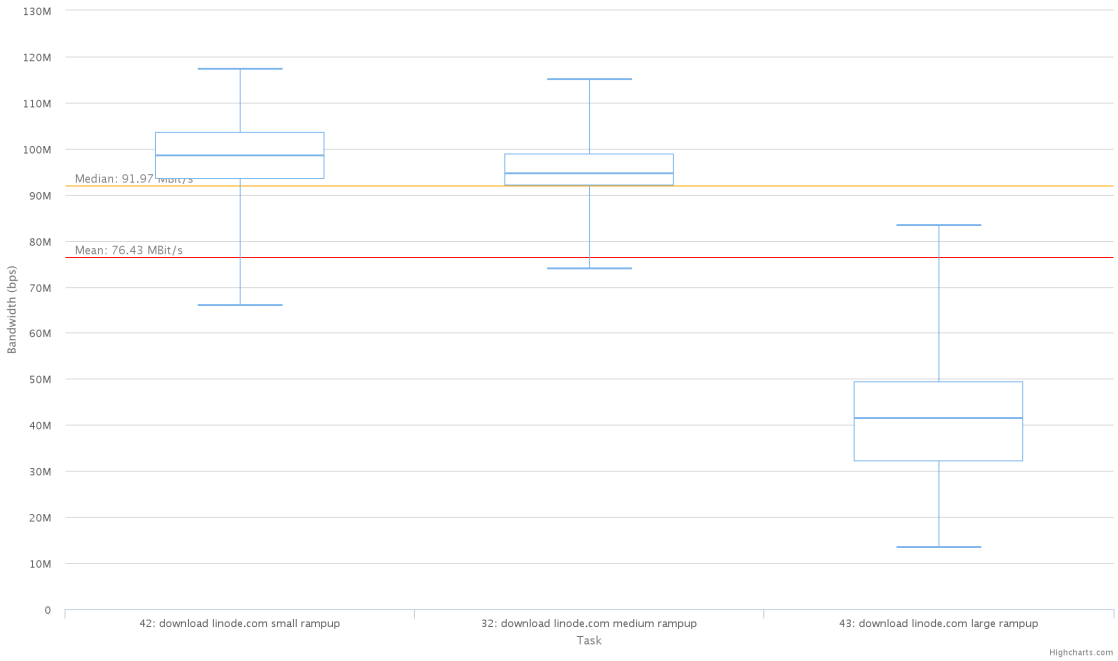


Figure 6.6: Box-plot for the (inconsistent) results of the ramp-up time parameter comparison for a 100 MBit/s probe

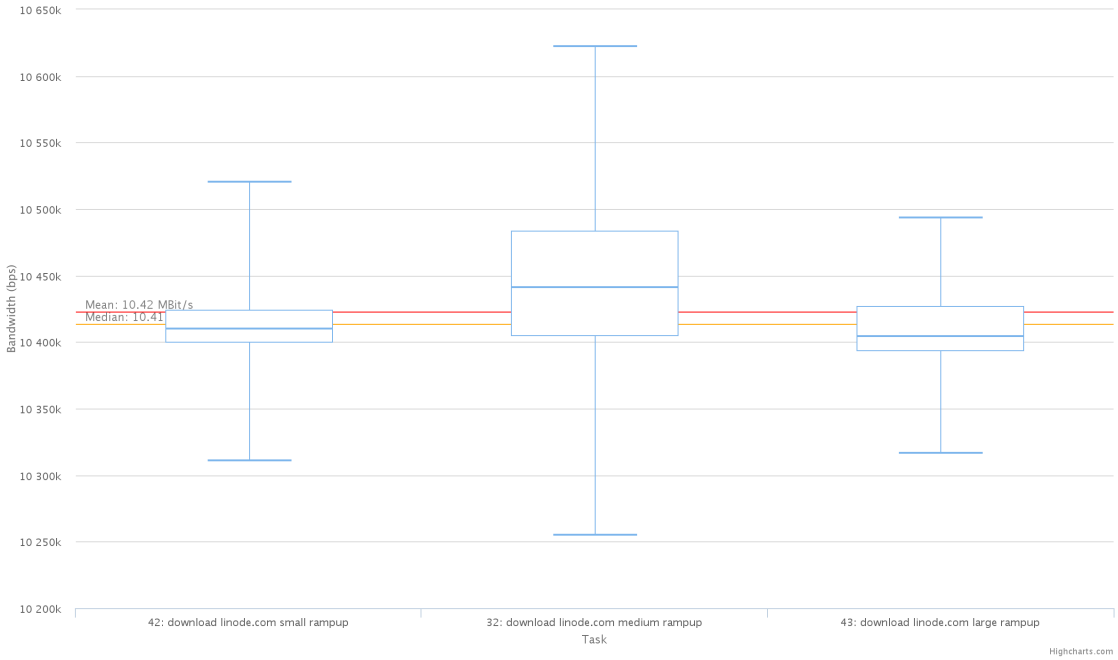


Figure 6.7: Box-plot for the results of the ramp-up time parameter comparison for a 10 MBit/s probe

connections are not enough to accurately determine the available bandwidth of the

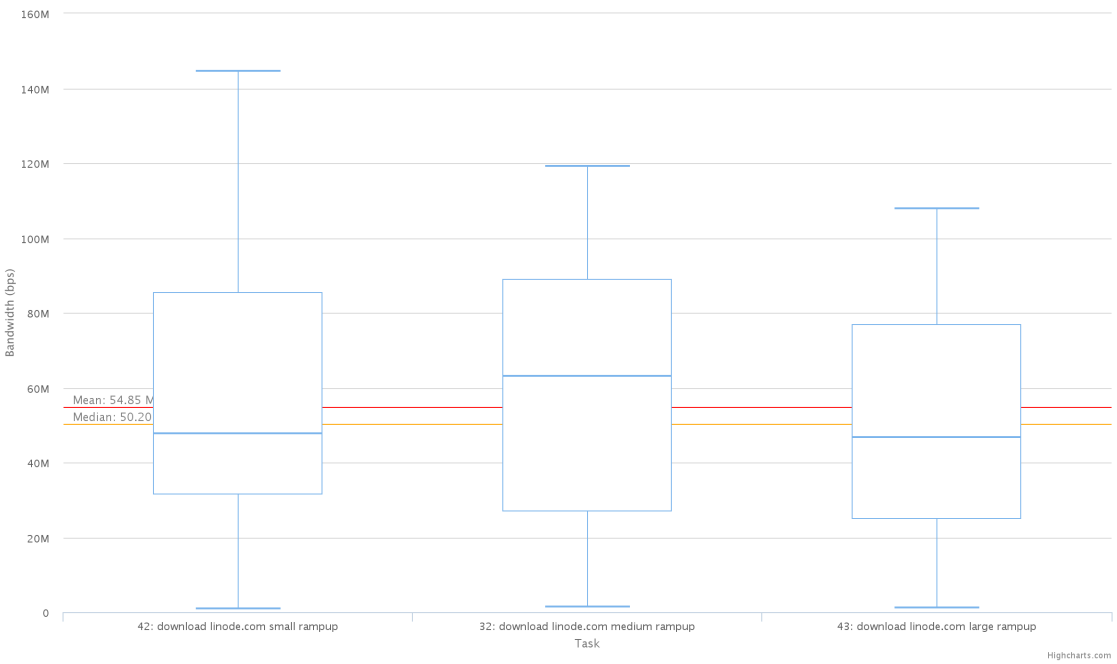


Figure 6.8: Box-plot for the results of the ramp-up time parameter comparison

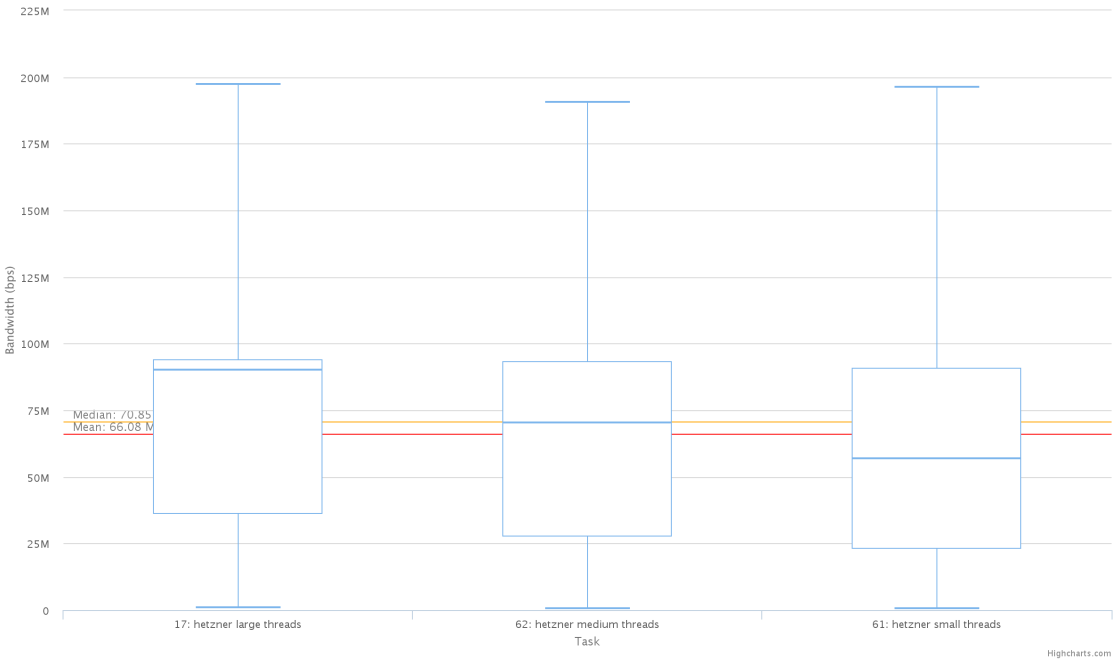


Figure 6.9: Box-plot for the results of the thread count parameter comparison

connection. With four threads, the results are stable and reflect the actual speed of the known probes. This is in line with the findings of Rufini et al. who describe that multiple TCP flows are required to reliably obtain the line capacity [RTMM14].

In their work they use five and ten TCP flows. Due to hardware limitations of the probes that were used for these tests, the GLIMPSE platform used no more than four threads for this measurement campaign. This is also consistent with the maximum number of threads used by Speedtest.net. It remains future work to increase the number of threads and compare the results to the findings presented here.

6.2.2 Destination Ports

In today's Internet, a number of middleboxes such as firewalls or performance enhancing proxies are deployed to rate limit or accelerate communication on certain well-know ports. To determine if the measurement results change when using a different destination port for the same server to download from, this measurement campaign scheduled six measurements using the following ports:

- 80 (HTTP)
- 8080 (often used as an alternative web server port)
- 5060 (port for the Session Initiation Protocol)
- 22001 (random registered port)
- 27015 (used for servers of some computer games)
- 52001 (random private or dynamic port)

The server used for all downloads was located in a data center of the Hetzner Online GmbH in Germany and had a 1 GBit/s Ethernet interface. The overall bandwidth available at the data center location was 1.15 TBit/s, while the guaranteed bandwidth to the server was 200 MBit/s. Measurement parameters were set to use 4 threads to download for 5 seconds with a ramp-up time of 2 seconds. The file used for all download measurements was a file generated with random data with the size of 268435456 bytes (which equals to 256 megabytes). As 4 threads are used in parallel to download this file this should be enough to support Internet connections up to approximately 1 GBit/s (the fastest downlink Internet connection at the probe locations in this test however only had a speed of about 250 MBit/s). nginx [ngi] was used as the web server with the file permanently loaded into the memory of the server.

Two campaigns were carried out for this experiment. The first one had all active devices participating for about two weeks, the second one used only two devices (the same as for the Speedtest.net measurement campaign described in 6.1) for three weeks. The box-plot in figure 6.10 for the first campaign with 12 devices does not reveal many interesting properties, as the results are very similar for all ports showing no hints for performance enhancing proxies or traffic shaping. Figure 6.11

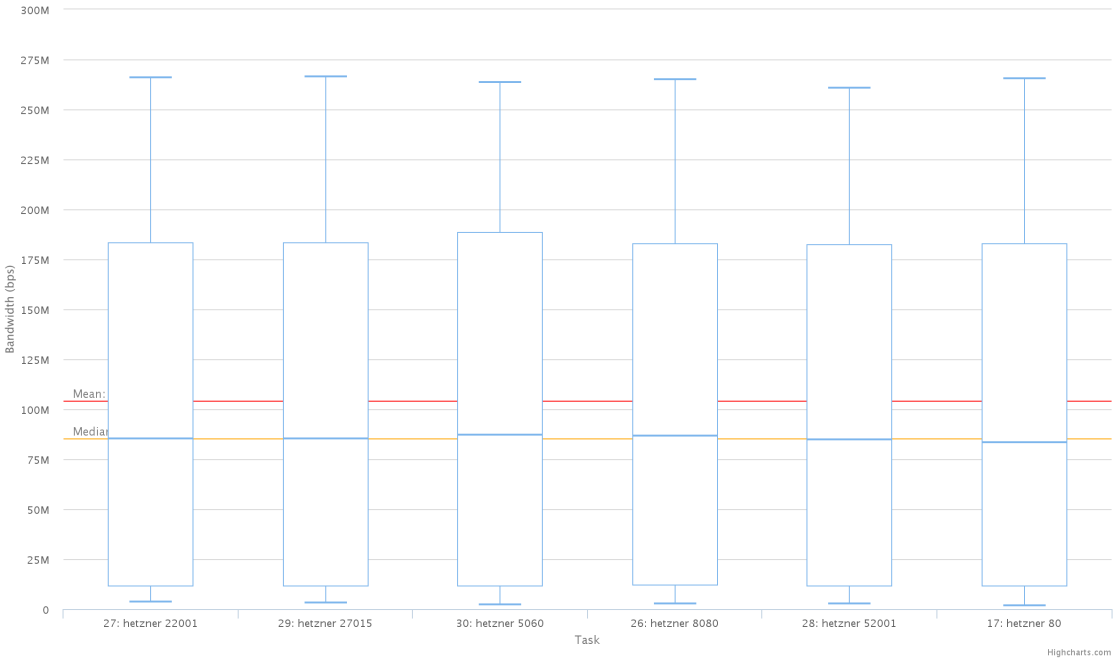


Figure 6.10: Box-plot for the results of the port measurement campaign

with the scatter-plot of the same data makes the different connection speeds visible by looking at grouped dots with the same speed. There are obviously devices with approximately 250, 175, 100, 50 and 10 MBit/s connections present.

Looking at all devices at once makes comparing the results rather difficult. Also, the basic conditions like available bandwidth, connection type (WiFi, Ethernet, Cable, ...) were not known for every device. Therefore, the second campaign with the already known residential networks was conducted to analyze the downloads with different ports in detail.

As mentioned before, both devices at these locations had 100 MBit/s available. As a reminder, location A showed inconsistencies while automatically selecting a download server by Speedtest.net, while location B showed higher and more inconsistent latency results. Figure 6.12 and 6.13 show the charts for this measurement campaign beginning with location B. Both plots do not reveal any particularities

when using different ports on this Internet connection. On the contrary, they show a nice and consistent available downlink rate over the whole measurement period.

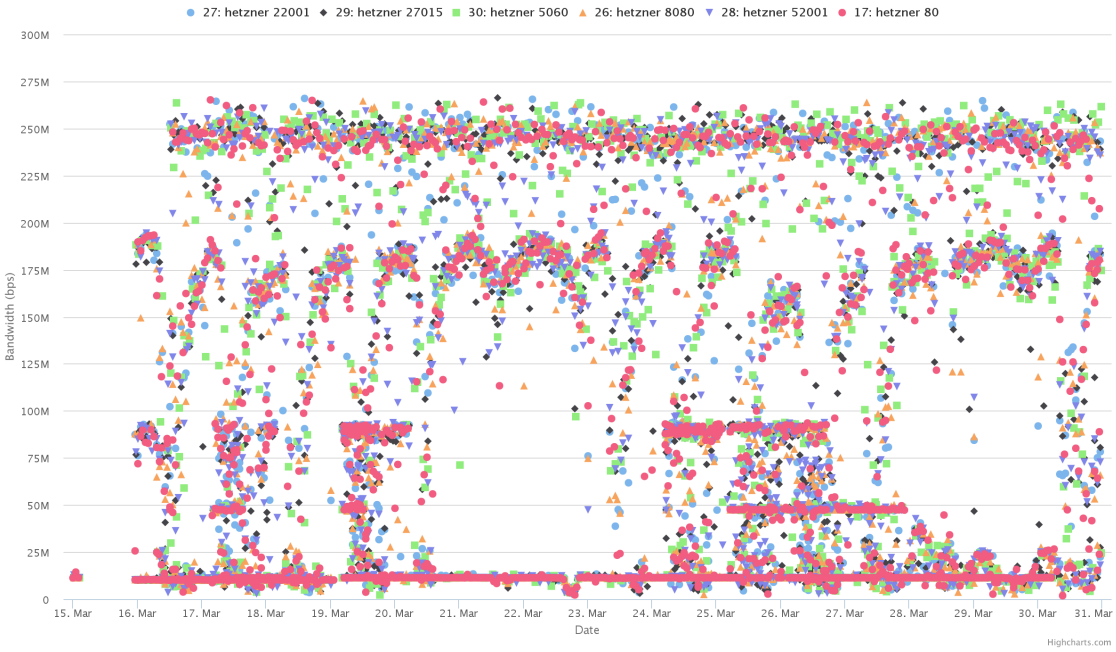


Figure 6.11: Scatter-plot of the results of the port measurement campaign

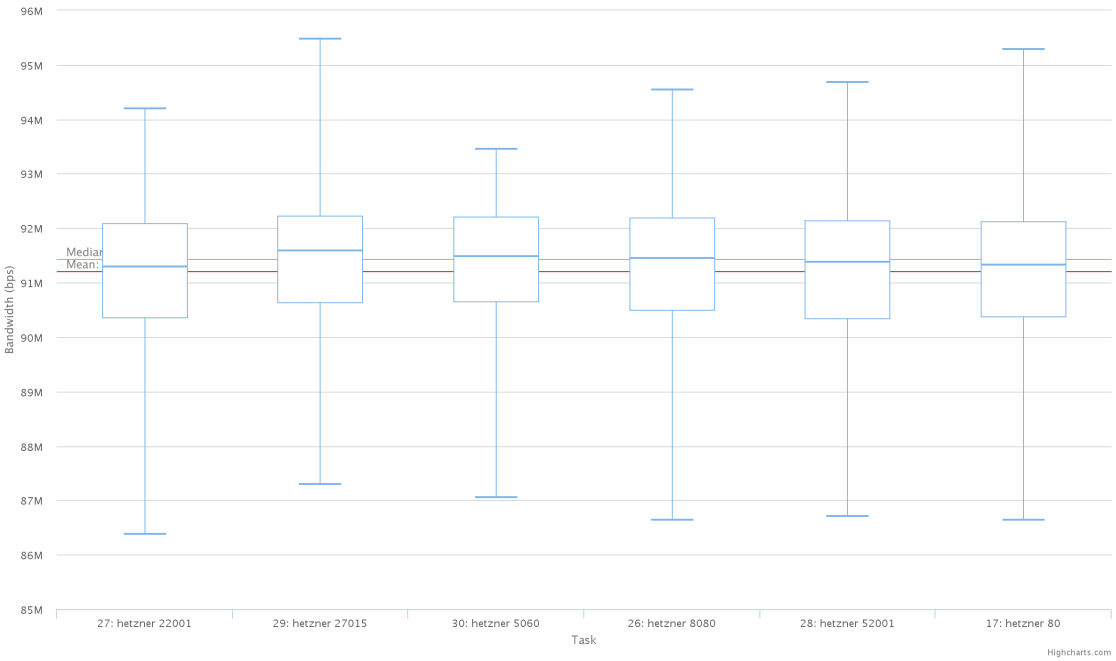


Figure 6.12: Box-plot of the results of the port measurement campaign in location B

The scatter-plot demonstrates vividly what should be expected from a connection without prioritization or throtteling of the tested ports.

The results for location A on the other hand show something unexpected. For once, measurements to port 5060 seemed to be blocked by the ISP. No results could be generated at all when trying to download using this port. Also, the variation of the measured speeds is way larger than for location B, the average (mean) bandwidth for all downloads was only at 64.23 MBit/s. As a control-measurement a file from a Speedtest.net server was downloaded in the same measurement period using GLIMPSE probes instead of the Speedtest.net Flash plugin. Figure 6.14 containing the box-plot for this shows that the results for the Hetzner-server were consistently bad, while the Speedtest.net server lead to very good results. The scatter-plot in figure 6.15 shows this even more drastically (and also reveals that there are some gaps where no measurements could be executed at all, the reason for this remains unknown). The same results could be reproduced with other servers, only Speedtest.net servers consistently generated good results. This could indicate that this provider prioritizes traffic for Speedtest.net to improve the measurement results of the users. An ISP clearly has an incentive to identify traffic caused by Speedest.net and prioritize it, as many users use Speedtest.net to estimate an ISP's service quality. This however raises serious questions about net neutrality, legality and business ethics. It is also possible that the peering-agreements of this (considerably small) ISP are in favor of Speedtest.net, while all other servers available to test against are routed through a bottleneck. Traceroute measurements do however not support this.

Looking back at figure 6.11, another anomaly is visible which has nothing to do with the destination ports used. Multiple results seem to have a trend to get better throughout each day from low two-digit downlink rates up to 200 MBit/s. The device responsible for this is a probe located in the network of the University of Applied Sciences Augsburg. This network has a download bandwidth of 200 MBit/s, but shares its connection with all students, members and servers of the University. Figure 6.16 shows the scatter-plot only for this device in the measurement period. It is clearly visible that the measured available bandwidth is low at the beginning of the day and gets better throughout the day with a peak at night-times. The weekends on the $21^{st}/22^{nd}$ and $29^{th}/30^{th}$ show much better results, therefore clearly indicating that the cross traffic present in the network while measuring is the cause for the measured bandwidth values. Of course this network with thousands of users is a very special probe-location compared to the other probes located in homes of end-

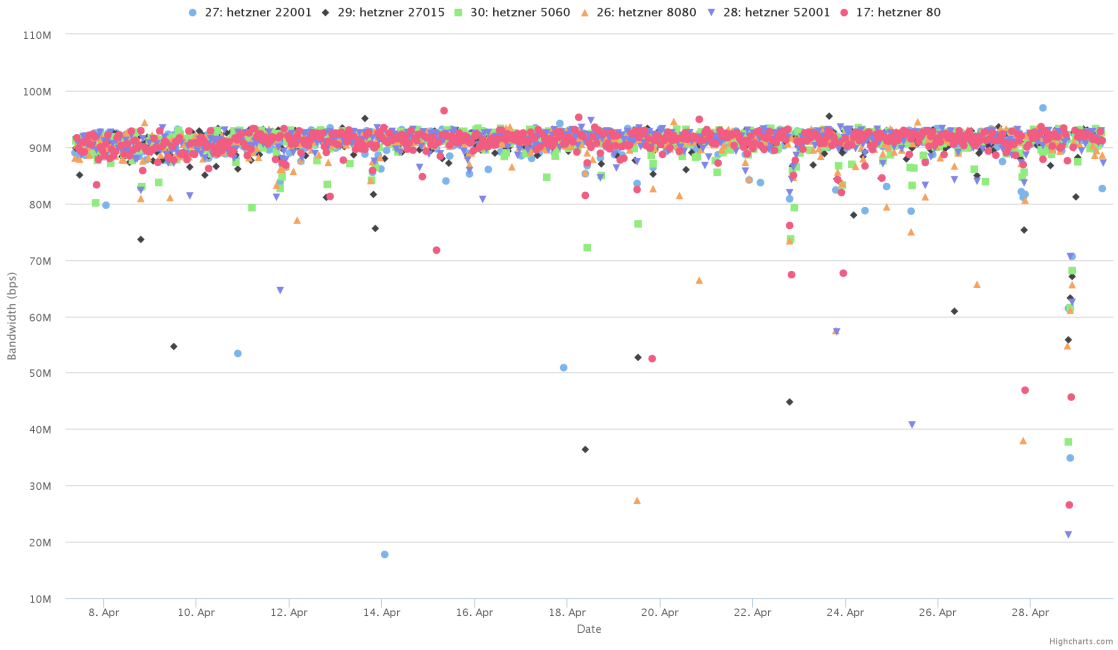


Figure 6.13: Scatter-plot of the results of the port measurement campaign in location B

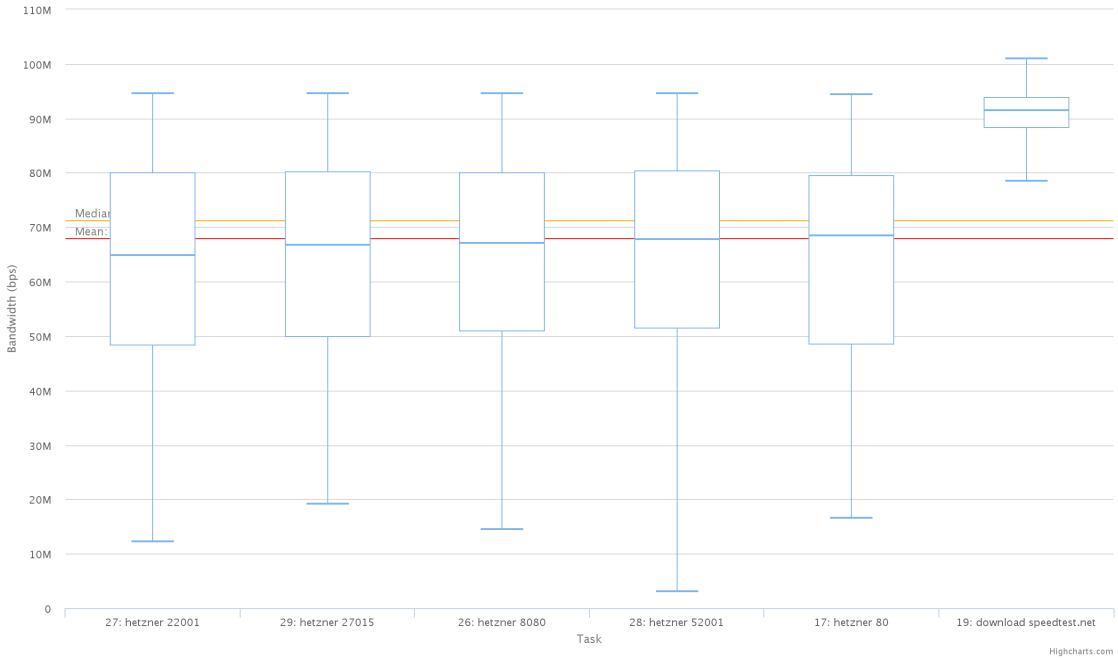


Figure 6.14: Box-plot of the results of the port measurement campaign in location A

users, but nevertheless is it important to remember cross traffic also when looking

at results from residential networks, as also in these networks, there are bandwidth-hungry applications that have a significant impact on such measurements (e.g. Netflix streams). Services like Speedtest.net do not have the ability to check for cross traffic. GLIMPSE however can identify cross traffic under certain circumstances as described before.

6.2.3 Lineode.com

While conducting some download measurements against different cloud hosting services, an interesting observation was captured. One measurement used a publicly available file from a Lineode server located in Frankfurt, Germany [Linb]. This file is explicitly provided for the usage in speed tests to determine the best location to host services or content in the Linode network [Lina]. Eight devices executed measurements using this file in a one week period. Some of them experienced consistent download speeds while for others, the speed drastically changed on the 19th of September 2015. Figure 6.17 shows the box-plot for one such device (device ID 20) for which the speed improved a little bit on the 19th and was considerably better on the 20th. Table 6.3 summarizes the values for the participating devices before, on

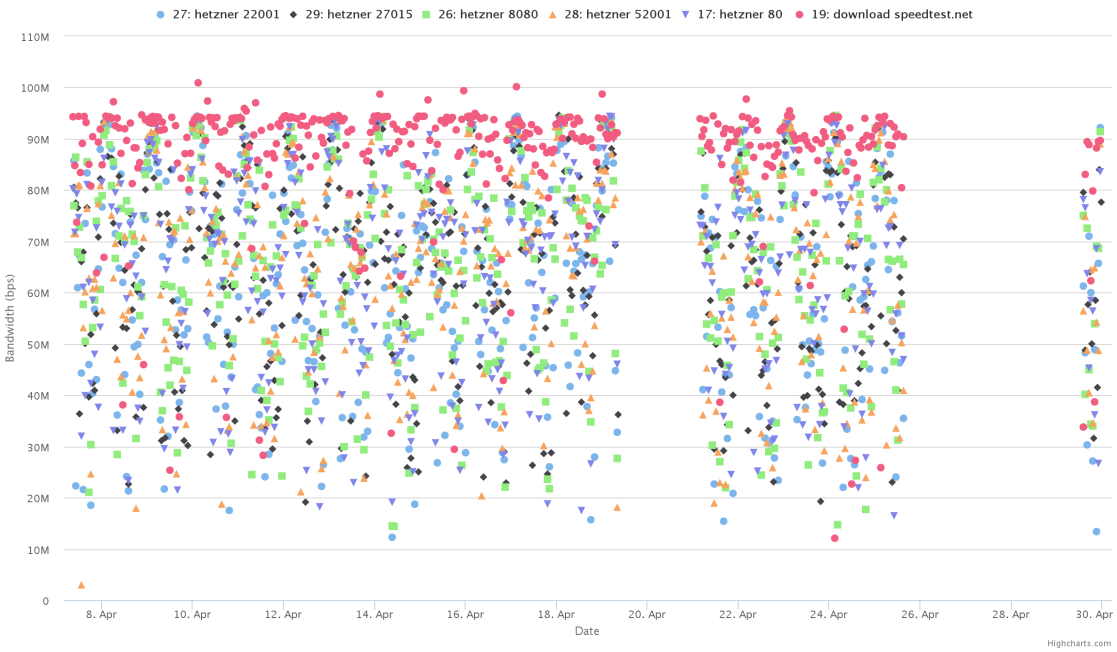


Figure 6.15: Scatter-plot of the results of the port measurement campaign in location A

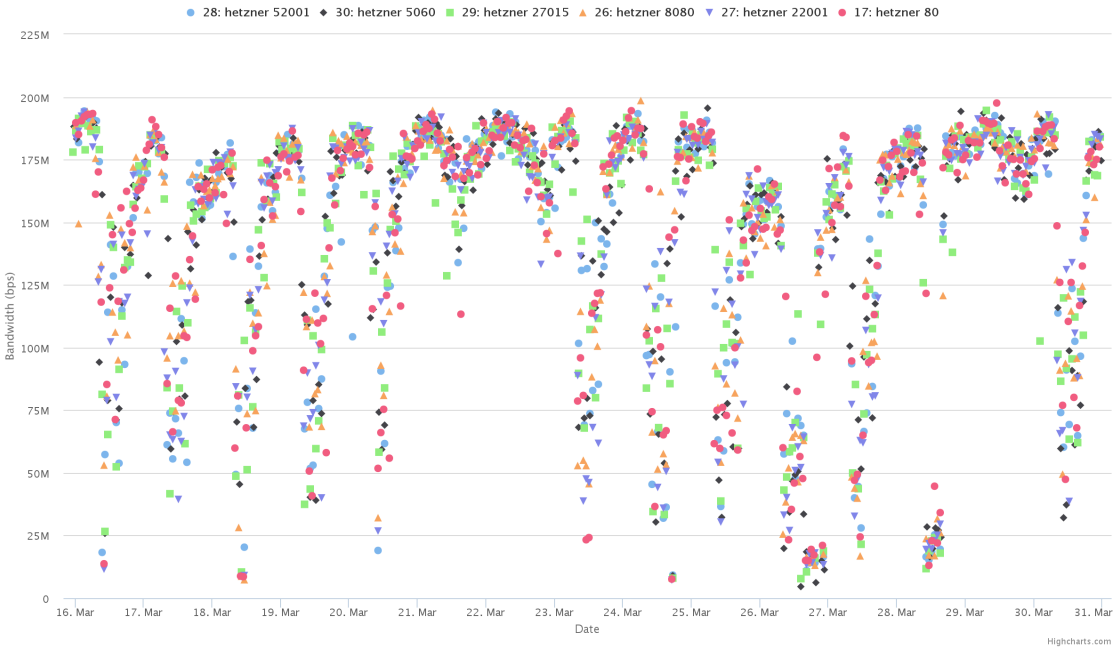


Figure 6.16: Scatter-plot of the results of the port measurement campaign in a universities network

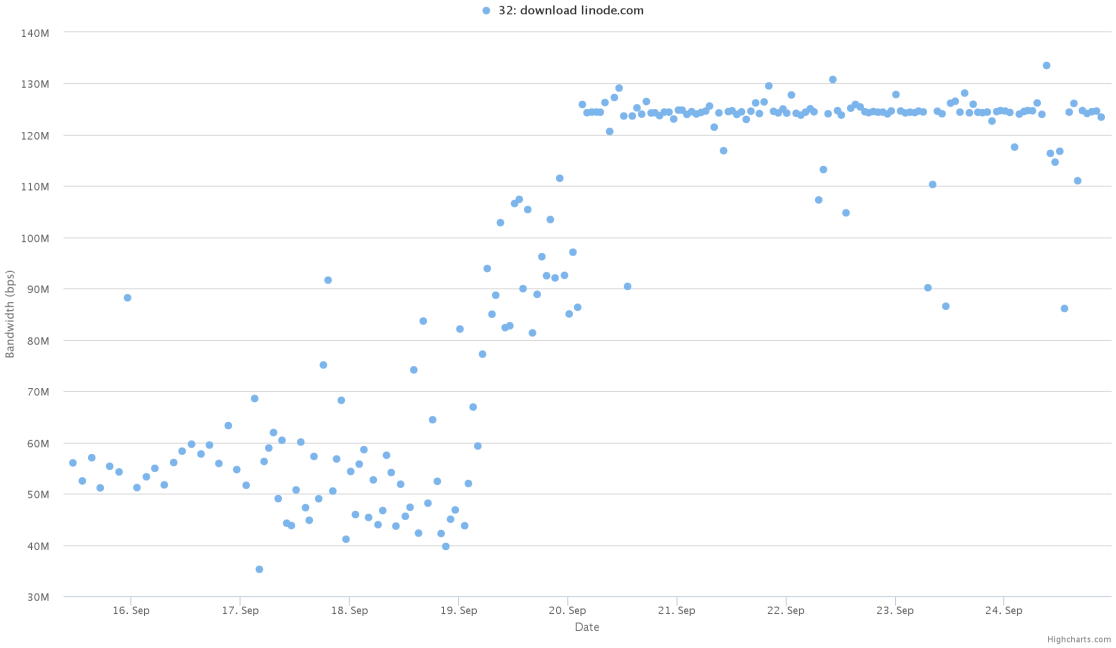


Figure 6.17: Scatter-plot of the results of the Lineode measurement campaign (device ID 20)

and after the date on which something in the Lineode network or on the Lineode

server changed[1]. For four devices the speeds got better (for most considerably better),

Device ID	Before 19th	On 19th	After 19th	Result
20	52.16	82.84	116.09	better
68	49.56	56.15	48.87	same
69	8.18	6.49	9.59	better
71	42.74	45.63	45.90	same
80	55.97	80.96	86.22	better
81	10.54	10.51	10.67	same
120	89.15	40.68	27.60	worse
144	51.59	66.31	84.71	better

Table 6.3: Average download speed in MBit/s from a Lineode server

for three it stayed the same (even though one of them got higher speeds on the 19th as before and after that day) and for one the speed drastically worsened. One device could even reach over double the speed than before.

The cause for this behavior and the underlying reasons remain unknown. Possible explanations might be a change in the routing path to the network, an update of server or infrastructure hardware or reconfigurations (e.g. load-balancing policies or fixing a misconfiguration). These are indeed probable causes, as the data center location in Frankfurt was new at the time. This measurement also nicely shows, that often longer lasting measurement campaigns are necessary to observe interesting changes in network behavior. This particular measurement however causes a large volume of traffic.

6.2.4 Server Location

Through a research cooperation with a regional ISP, a server could be deployed in the core network of this ISP and used as a target for GLIMPSE measurements. The server was located very close to the Broadband Remote Access Server (BRAS), which allows to nicely isolate effects measured on the access part of the ISP's network. To determine if and how the server location influences the measurement results, this campaign executed multiple HTTP download measurements from a probe using

[1]The previously used devices in residential networks called location A and B from the Speedtest.net and port measurements have the device IDs 80 and 68.

the DSL access network of this ISP to the server described above (which resided in Munich, about 70 kilometers away from the probe), to the previously used Hetzner-server (which was located within 200 kilometers of the probe), to a server of the university available at measure-it.net (located in the same city as the probe) and a publicly available server from Speedtest.net (located within 100 kilometers of the probe). The measurement period was about ten days, the resulting box-plot can be seen in figure 6.18. The results for the nearest server are the best and get worse the

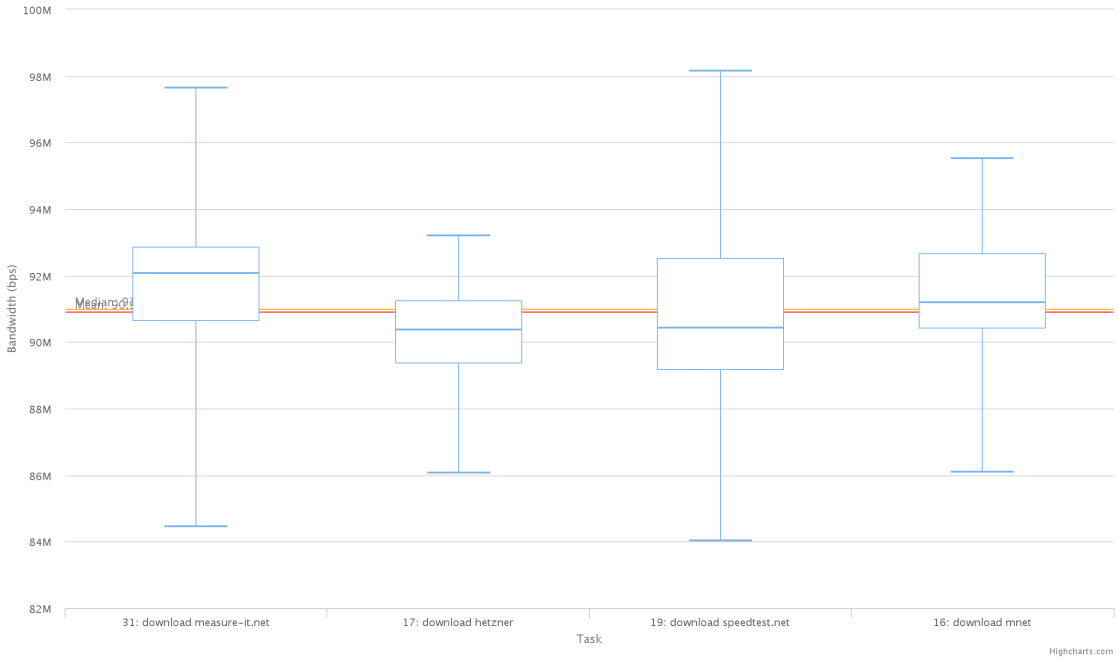


Figure 6.18: Scatter-plot of the results of the server location measurement campaign for one probe

farther the server is located from the probe. All average values are however within 1.5 Mbit/s of each other and can therefore all considered be good and trust-able results. The topological distance (in terms of IP hops) to all locations is very close and the variations of a few percent can be to many factors such as cross traffic.

6.3 Latency

To determine the accuracy of the implemented latency measurement using UDP packets a measurement campaign was done using the GLIMPSE latency measurement to `www.facebook.com`, `www.audi.com` (hosted on the Akamai CDN) and

`www.google.com` with both the system-ping type and the UDP-ping type (as described before in section 5.3.1).

First of all, the resulting box-plot for all participating probes can be seen in figure 6.19. Drawing a scatter-plot for all probes was not possible due to the number of

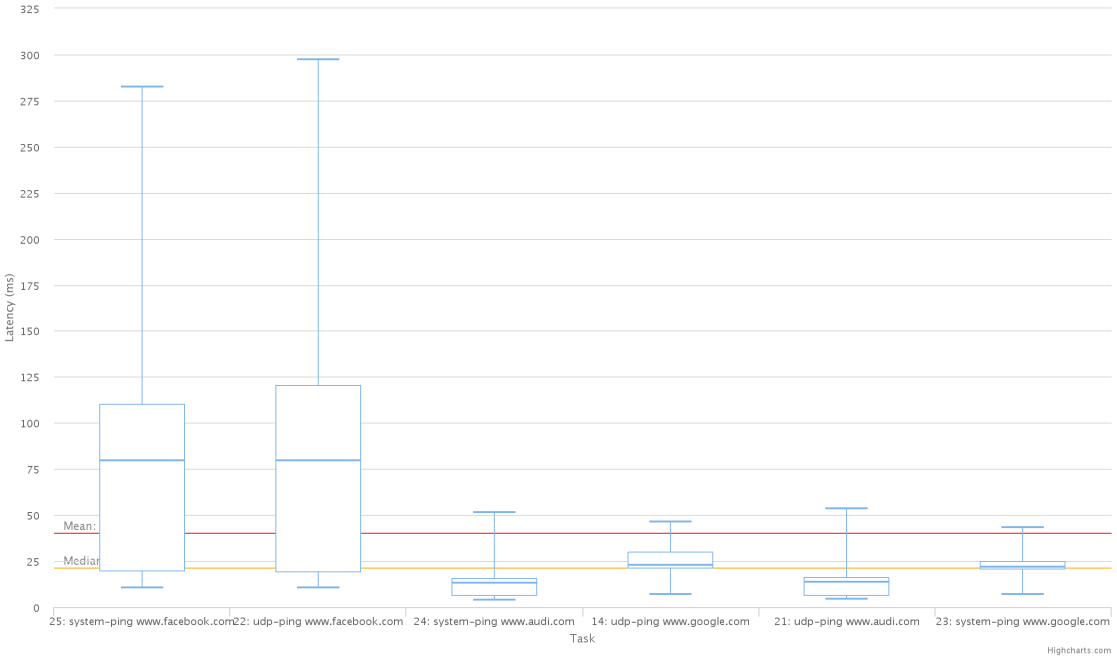


Figure 6.19: Box-plot of the results of the latency measurement

data-points being to high to be displayed with the used visualization method in the GLIMPSE dashboard. Overall, 30,506 latency measurements were conducted by 17 probes within a period of one month.

For `www.facebook.com` the chart shows almost the same mean latency: 79.80 ms for the system-ping and 79.76 ms for the UDP-ping, but the latter shows a larger variation and more outliers. The in general much higher latencies for `www.facebook.com` can be explained by the fact that packets travel through servers in the US while the destinations for the other domains are located in Germany. For `www.audi.com` (`www.google.com`) the system-ping resulted in slightly better (meaning faster) results with 13.30 ms to 13.59 ms (21.82 ms to 23.11 ms). All of these differences between the latency results for system-ping and UDP-ping are insignificant. Assuming that the system-ping implementation is (albeit being throttled occasionally) fairly accurately estimating the round-trip-time, the implemented UDP-ping can be regarded as accurate. It might be that at devices that implement ICMP throttling (e.g. sometimes routers throttle or delay answers to ICMP messages), the UDP-ping implementation

provides better results. This however is for further study.

7 How Broadcast Data Reveals Your Identity and Social Graph

For various reasons described previously, GLIMPSE was restricted to perform active measurements only. These reasons were mainly user acceptance, privacy concerns and access rights. But from a research perspective, end devices and the traffic these devices send and receive are clearly of great interest. Therefore, this "what if" study was performed to see what interesting research questions could be answered if GLIMPSE were able to passively observe traffic. As the Internet is moving steadily to an all encrypted world where analysis of even passively collected data is difficult, a study on broadcast and multicast data was performed.

This work was published at the International Wireless Communications & Mobile Computing Conference 2016 as part of the 7$^{\text{th}}$ International Workshop on Traffic Analysis and Characterization and received the best paper award [FWW16].

7.1 Abstract

Networks rely on broadcasts and multicasts for some of the most basic services such as auto-configuration. In the recent past, application layer protocols have increasingly made use of the broadcast mechanism. Examples of these applications include Dropbox, Spotify or BitTorrent Sync. Given that broadcasts can be seen by every device in a broadcast domain, information that can be gleaned from this traffic is trivially accessible by a passive observer. Therefore, an obvious question is: what does broadcast and multicast traffic reveal about a device, a user or a group in a network?

To answer this question, the broadcast traffic of two fairly large wireless networks was analyzed. One of these networks was the campus network of a university which was analyzed for a period of six months. Also, two SSIDs of the IETF meeting network in Yokohama in November 2015 were analyzed for a period of about 36

hours.

In addition to a general analysis of the composition of the daily broadcast traffic such as protocols observed, the number of devices, the peak times of user activity etc., a more in-depth analysis of a few protocols was carried out in order to identify users and their relation to each other. In other words, we used the available broadcast data to show that it is possible to generate a social graph of the network's users base, which e.g. helps to identify groups among students, their course of study, their online times and other personal information. We have verified the correctness of our inferred social graph by asking students to confirm our findings.

None of the observed broadcast protocols alone is to blame for the above and there is no easy technical solution to the problem while retaining the benefits of the broadcast protocols. However, there is a simple yet effective countermeasure against this kind of analysis which is non-technical and "only" requires changing user behavior.

7.2 Introduction

As RFC 919 notes: "The use of broadcasts, especially on high-speed local area networks, is a good base for many applications" [Mog84]. While RFC 919 was written in 1984, it seems that many application developers have re-discovered the value of broadcast message exchange. Popular applications and services that utilize broadcasts today include Dropbox, Spotify, Steam and UPnP. Given that the used protocols all disclose information to every host on the subnet, it is trivial to collect this information. Protocol designers have certainly made sure that when presented with their protocol's information alone, a passive observer will not be able to make any good use of it. However, since other applications also broadcast information, these data sources can be combined to learn about devices, users and groups of users on the network.

In order to analyze what information today's broadcast traffic gives away, we have looked at all broadcasts sent on a large wireless campus network serving thousands of users at our university from October 2014 to April 2015 (note that we have analyzed both broadcast and multicast messages but will only refer to it as broadcast in the text to make it more readable). In addition, we have also analyzed the broadcast traffic on the IETF meeting network during the 94^{th} meeting in Yokohama in November 2015 as a second data point to demonstrate that this analysis can be

done at a public event. Due to restrictions of the IETF network and the much briefer measurement period the focus of this work is on the first data point.

7.2.1 Methodology

We performed both experiments by connecting a computer to the network like any user would do, therefore we observed only traffic accessible by any participant of the network. The broadcast traffic from all users in the university's network was visible when connecting to one access point, therefore only one capture point was necessary to gather all data.

Our overall goal was to learn as much as possible not only about the devices that send out the broadcast messages, but also about their users and the social graph of these users relying exclusively on data that any user of the network could access. We used insider knowledge only to verify the correctness of our deductions.

7.2.2 Motivation

The idea for this research formed while visiting a conference. To access a private network of a university a VPN tunnel was used over a mobile LTE connection. As usual for mobile data plans, the amount of traffic used over the LTE connection is limited. After closing most of the running processes on the used computer in order to minimize the data consumption, traffic counters still showed a constant stream of incoming packets with approximately 150 kbit/s. Further investigation with Wireshark [Wir] showed that most of these packets were broadcasts, which lead to the question which protocols and devices introduce this traffic and what to learn from this data.

7.3 Ethics and Anonymization

Tapping in on traffic on a network generally raises privacy concerns, even if it is broadcast traffic, which means data is trivially accessible by everyone on the subnet and the listener is actually part of the intended recipient group. Local legislation might or might not allow this kind of analysis and can require certain procedures and precautions.

Therefore, before we started our large-scale experiment, we used our lab to sample broadcast and multicast data. During these experiments it became apparent that

personally identifiable information (PII) can appear in some of these messages. This required us to use anonymization techniques on the data beyond securely hashing information such as MAC and IP addresses. In particular, real names were frequently found in hostnames. We therefore tokenized hostnames found in all protocols sending them, removed tokens from a list of well-known terms as explained later and hashed all remaining tokens which potentially could contain PII. This removed PII from the collected data and made the later analysis much more difficult.

We consulted with the legal department of the German research network (DFN) and received a detailed statement from them. Meanwhile, they have published their work on this particular legal topic in [Spo15]. We believe, based on this statement, our experiment conforms to German data privacy law and user privacy was protected during these experiments. Also, for the experiments on the IETF meeting network, we have consulted with a number of experts in the field, parties familiar with local law and the IETF leadership, in particular the chair of the IETF. The experiment was announced to the participants of the meeting and only a selection of the available SSIDs announced during the meeting were analyzed to give meeting participants the opportunity to opt out of the experiments. There was a lively discussion on the meeting mailing list following the announcement of the experiment [att15a]. We actually tried to do this experiment at the meeting 93 in Prague, but we were not able to receive a definitive answer on some of the legal questions in time [att15c]. Also, at that time there was a very lively discussion on the privacy implications of such an experiment [att15b].

Given the concerns expressed whenever our experiments were announced, we believe it is important to analyze current broadcast protocols in a privacy-preserving manner, to have a more informed debate about the issue and to understand what broadcast data currently reveals in terms of sensitive information. In particular since the data collection we performed can be trivially done by an attacker just by passively listening, without having to have special privileges or having access to a special location in the network.

7.4 Data Analysis

We base most of our analysis on the larger data set collected at the university. There are two reasons for this. For one, given the larger time window of the analysis, the results are more expressive. Additionally, on the IETF network, broadcasts were

not distributed over the air interface and only multicast messages were seen on the network. Therefore, when the data set is not explicitly identified below, we will be talking about the university's data. We will explicitly mention when the IETF data is being addressed. The smaller data set is nevertheless quite valuable since represents an interesting second data point with a very different user base—highly knowledgeable domain experts.

On the university network, the total volume of broadcast data seen during the six month time period did amount to about 40 GB, counting every byte starting with the Ethernet header. Broadcast data peaked at around 1 GB per day with a low of 3.5 MB and an average of 215 MB per day. Analyzing this data—even live—does not require special hardware and can clearly be done using a low-performance device. Figure 7.1 shows the traffic per day for the measurement period. As expected for a university network, most traffic could be seen on weekdays during term time, while the weekends and non-term days showed far less activity.

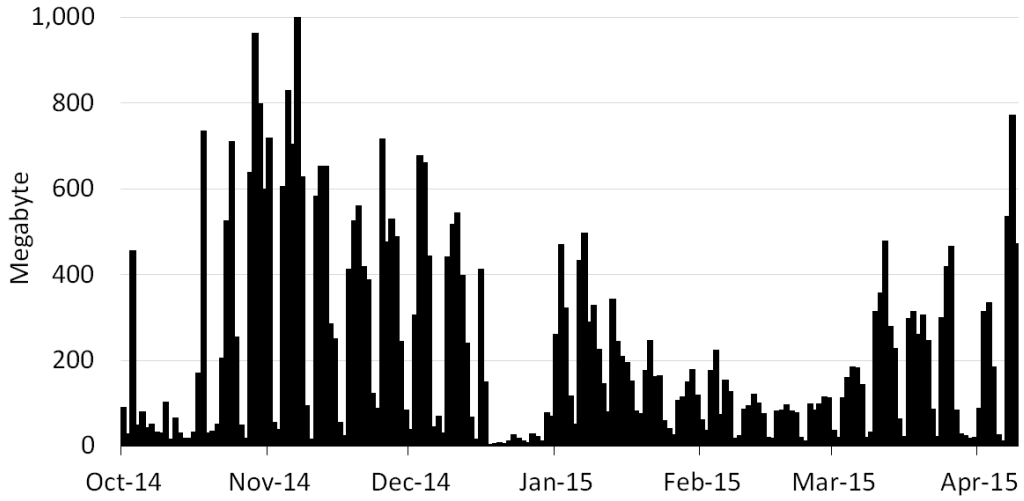


Figure 7.1: Traffic per day (university network)

The data contained about 35,000 different MAC addresses of which we suspect a maximum of 21,000 to be from real devices. The other MAC addresses were either from apparent experiments by students or from VPN clients that were part of the same broadcast domain for about half the time of our data collection. Luckily, it was fairly easy to identify these and to remove them from the data set.

Of all the packets observed, approximately 90% were UDP packets. To find the protocols worth analyzing, we looked at which protocols accounted for the most UDP packets sent. They were: mDNS (16%), SSDP (15%), LLMNR (13%), NetBIOS (7%)

and Dropbox LAN Sync Discovery Protocol (7%). We focussed our attention mainly on mDNS, NetBIOS and Dropbox after inspecting the information these protocols could reveal. Including the other protocols as part of this analysis remains future work.

The hourly multicast traffic of the IETF network (both monitored SSIDs combined) in the 36 hour measurement window can be seen in figure 7.2 and did amount to nearly one GB. During that time, about 2,600 MAC addresses were observed. Out of the protocols above, the largest fraction of the multicast traffic consisted of mDNS as well (36%)—remember that no broadcast traffic was recorded on the IETF network and therefore some protocols were missing.

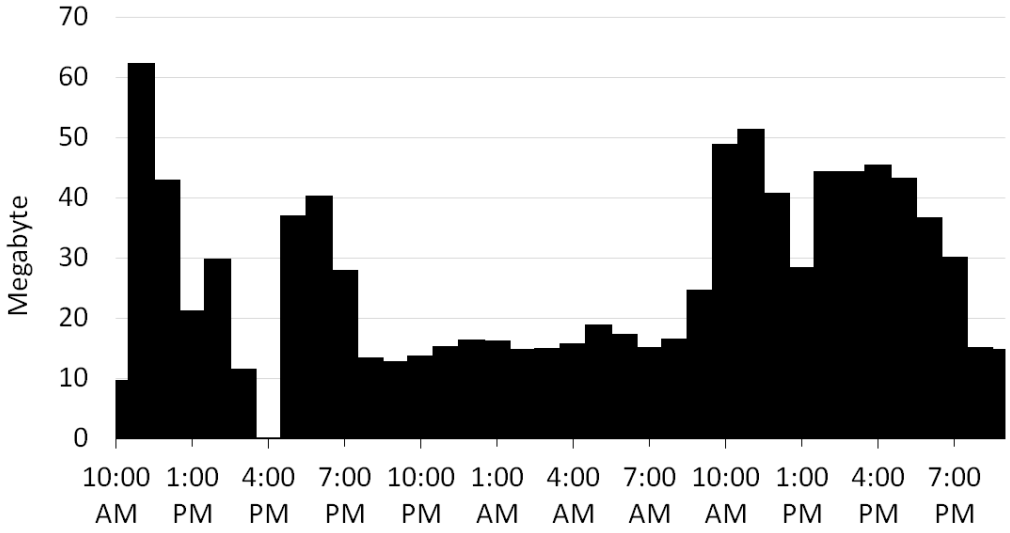


Figure 7.2: Traffic per hour (IETF meeting network)

7.4.1 Dropbox

The Dropbox desktop application uses the Dropbox LAN Sync Discovery Protocol to speed up the synchronization of shared directories that are present within the local network. Such folders do not have to be synced from the Dropbox servers but from other Dropbox users nearby to save bandwidth. This option is enabled by default and leads to multiple broadcast packets on UDP port 17500 every 30 seconds. The datagrams sent contain a unique identifier for the Dropbox installation of one user, the so-called *host_int*, which makes it possible to track this user even if the IP address or MAC address changes. This also enables us to identify two interfaces, e.g. a WiFi and a LAN interface, to belong to one device. The broadcast packets

also contain *namespaces*, which is a list of unique IDs for the shares this user has. In other words: if two users share one directory, they announce the same ID in the namespace list.

For the six month period, 2,560 Dropbox user installations with 9,361 shares overall could be observed. The users of the analyzed network are mostly students, thus we suspected that many of them would use the shares for specific lectures or the whole course of studies, meaning they would show up as a community if we drew the Dropbox users and shares as a graph. As the complete data covers two semesters with vacation time in between, we decided to analyse only the Dropbox data from the first three weeks of the summer semester beginning on the 16$^{\text{th}}$ of March 2015. We removed all shares which showed up in only one namespace list as they are not of interest for an analysis of the social graph of the network's user base, i.e. these shares might have multiple users but only one within our network. After that, we removed the users which only had such a share. This left us with 712 users and 718 shares, for which we drew a directed graph. This graph is an abstract representation of a social graph between the users of the network. It shows who is sharing data with whom, but it does not reveal who these users are, which course they attend or why they have a connection.

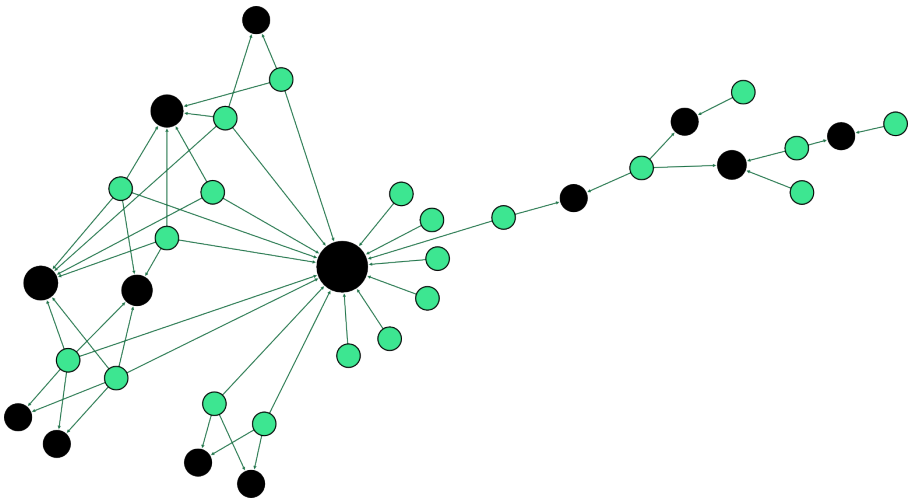


Figure 7.3: Dropbox community graph

In order to group users with stronger ties into groups, we applied the Louvain method to identify 101 communities within this graph [BGLM08]. The largest of those had 48 users and 31 shares (with 42 users for the largest share). We chose six communities to investigate further for our analysis based on the number of users,

shares and modularity. One of those communities is shown in figure 7.3. The black nodes represent shares, the other nodes represent users connected to the shares they announce. We numbered all communities and use those numbers throughout this chapter to identify them. The graph in figure 7.3 shows community 54.

At this point we tried to identify publicly available information that could help us to further enrich those communities with information. We e.g. crawled the university's publicly accessible course schedules and tried to match online times of community members and course schedule times. Unfortunately, this did not yield good results in general.

To match these communities to the courses of studies at our university, we compared the online times of the community members to the lecture schedules. The University of Applied Sciences Augsburg has a publicly accessible website for all schedules for the different semesters, which also offers an iCal version of each schedule. We wrote a script to crawl those and extract the start and end times of each lecture in each course of studies for the three weeks. The online time of the community members could be easily extracted from the data as the Dropbox client broadcasts the *host_int* with a high frequency. Even when the Dropbox client was off, we still had the *host_int* to MAC address mapping which helped tracking the online times of users. However, far too many courses of studies matched the online times of the communities. It turns out that most of the faculties at our university add multiple timeslots for group projects and other placeholder entries into the online schedule. Some faculties still announce the real lecture schedules on a blackboard. This means in our case that it is impossible to identify the course of studies with only the online times of each member of a community, therefore we had to look elsewhere for more information about each member of the group in the data.

7.4.2 Hostnames

The abstract social graph the Dropbox protocol reveals has interesting properties in itself. E.g. one can see how interconnected students are. But to make it really interesting, the nodes in the graph need to be correlated with actual user names. This clearly is information that no sane protocol designer would broadcast in the clear. A number of protocols however broadcast the hostname of the device. NetBIOS over TCP/IP and mDNS are prime examples for such protocols, we analyzed and evaluated these.

Approximately 7,600 unique hostnames were announced from 10,500 different

MAC addresses. We processed these hostnames to remove duplicates (e.g. "John-Does-iPad" and "John-Does-iPad-2") and recurring strings such as "iphone", "macbook" and others. A little more than 5,300 hostnames remained for us to consider. To anonymize this list before analysis, we separated each hostname into tokens (e.g. "John-Doe" into "John" and "Doe") and stored them after hashing them securely first.

The most interesting finding in our analysis was that most users name their device after themselves using either their first, last or both names as part of the hostname. For our analysis, we only had hashed information available. In order to come to this conclusion, we needed a second data source from which we could extract names of people at the university, hash those too, and match for equality. This source is introduced in the next section. In addition, other interesting information can be extracted from the hostname, e.g. the language the user speaks. Apple users reveal particularly liberally their language by having names such as "iPhone von John Doe", where the "von" indicates a German-speaking user. A lot of hostnames reveal the device vendor, the type of device (phone, laptop etc.), the device model but also locations, faculties, functions (e.g. file server), or login names, initials and nicknames. We observed this information in control experiments conducted with a larger number of students who gave their consent to our experiment before connecting to an access point provided by us.

Enriching the social graph with this data helped identifying nodes in it. Also, mining information from social networks of well-known nodes like a list of friends can lead to the identification of previously only partially identified nodes, for example if only the first name is present in the hostname or initials. We did this manually for some members of our research group and students of the control experiments and were surprisingly successful. But to automate this for the gathered hostnames we needed to find another data source.

7.4.3 Additional Data Set

The LDAP server of the university is accessible from the campus network and LDAP searches are not restricted. This allowed us to automate the above process. We extracted all user data the server offered which included the login name[1], first and last name[1], email address[1], faculty, course of study, status (student, professor,

[1]This field was only stored hashed for our analysis.

etc.) and the date when the password has last been changed. We collected more than 8,400 user records this way. Out of those users, 1,300 had a unique first name, i.e. no other user had that particular first name. In contrast 4,564 last names were unique.

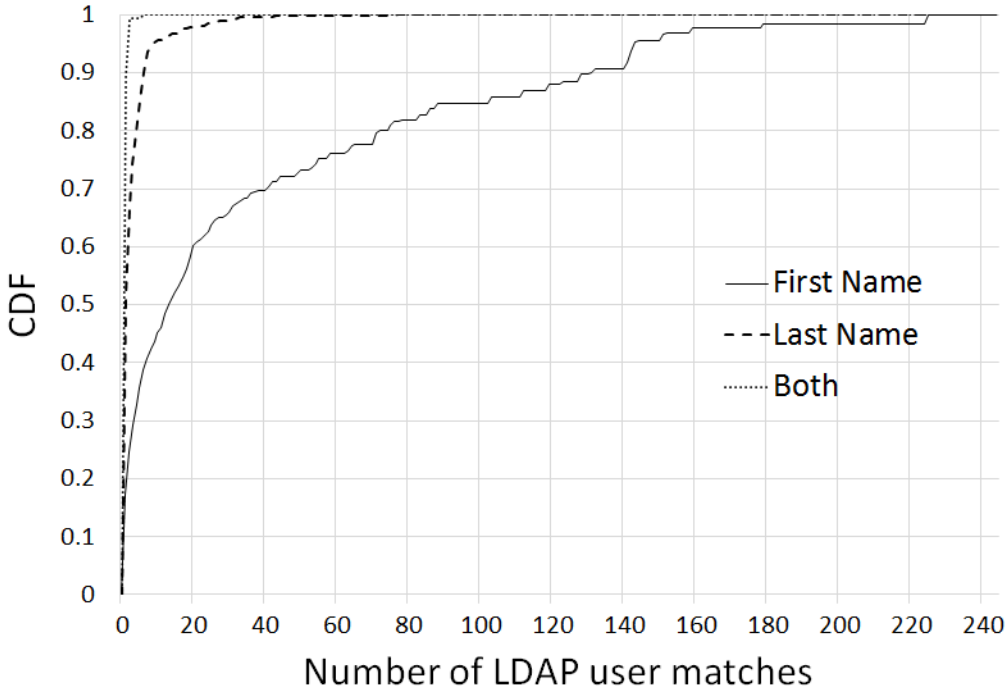


Figure 7.4: CDF for LDAP user matches based on the hostname

Using the list of 5,300 hostnames and the LDAP information, we searched for all hostnames that contained an existing first name, last name or both as part of the hostname. Interesting for us was whether the name itself would uniquely identify the user or how many LDAP users would remain as potential users of the device. As an example, if "John"(hashed) was part of the hostname, we counted the number of people called John in the LDAP data. If only a single John existed, we could identify the user uniquely. If five Johns existed, we could at least narrow the potential users down and combine it with e.g. the Dropbox data, online times, course of study of the host's Dropbox contacts etc. to finally identify the correct John.

We found approximately 2,900 first names as part of hostnames, where we did not count first names twice, e.g. "John-Does-iPhone" and "John-Mays-PC" would only count once as the occurrence of "John" in hostnames. Out of those, around 500 would uniquely identify the LDAP user. The CDF of name matches is shown in figure 7.4. 17% of the first names used in hostnames did uniquely identify the user

of a host. A quarter of the first names matched either uniquely or at most appeared twice. The biggest number of matches observed was 244, i.e. there is a first name used in hostnames that matches 244 LDAP users.

We also found 929 last names used as part of a hostname. The probability of uniquely identifying these users is—not surprisingly—much higher. Over 50% of the last names used as part of a hostname would uniquely identify the user of the host. The highest number of LDAP users that match a last name found as part of a hostname was 77.

Finally, in case a full name was part of a hostname, the probability for uniqueness is around 90%. 293 full names were used, which was still surprisingly high. Ambiguities were—of course—much rarer. But still, there were two full names that matched six LDAP users.

Since mDNS was the main contributor of this data, we were able to do this analysis also on the IETF network data set. We still needed a list of potential users of the network in order to create a list of hash values to compare against our list of hashed hostname tokens. LDAP was not available for this on the IETF network, but the attendees list is publicly available. We extracted first and last names from that list and applied the same method as described before. Overall, 1,487 people attended the meeting. On one SSID (SSID1) we observed 1,115 different MAC addresses and on the other (SSID2) we saw a total of 1,500 MAC addresses during the 36 hour measurement time window. The total volume of mDNS traffic was around 220 MB (SSID1) and 140 MB (SSID2). Of the hostnames found in the mDNS messages, on SSID1 6 (4) contained both first and last names, 98 (133) contained last names and 247 (195) contained first names of IETF participants for SSID1 (respectively SSID2). Of those 148 (98) would lead to a unique identification of the device owner.

The above was all done automatically using scripts, meaning some names are probably missed due to hostnames with initials or nicknames (e.g. “Bill” for “William”). These are easy to parse for a human but much more difficult to parse in software. With more effort, the above numbers can therefore be increased further. Our control experiments showed that our automated analysis on anonymized data indeed misses some names a human would easily identify as a nickname or similar. Also, without anonymization, a second data source such an LDAP server or the IETF attendees list is not necessary for a real attacker to infer identities.

7.4.4 Combining the Data

With the mapping of Dropbox users to LDAP users, we could attach (hashed) names to some of the nodes in our six chosen communities. For two of them, we could not identify any users as the hostnames only revealed ambiguous first names (community 36 and 56). In one community (19) we could identify two students, one used the full name and the other had a unique first name. These two users had the same course of studies, so we searched in our LDAP data for the other users in this community where we only had ambiguous first names from the hostnames. This lead to seven additional matches based on the course of studies and the time they had changed their passwords the last time. The LDAP server at our university does not return the date of creation or semester for a user. But the data showed most users do not change their password after they set it once (which probably is at the creation of the account), so we could calculate the semester with the help of the date of the last password change. Now that we knew the course of study and semester for this group, we could also verify that the online-times of the devices matched the semester's schedule after manually identifying lecture times where all students of the course should be online.

We re-did this for the other three remaining communities and decided we needed to verify the results of this analysis since we worked on anonymized data.

7.4.5 Data Verification

As mentioned before we did multiple control experiments with groups of students who consented to connect to an access point where we recorded and analyzed the broadcast data in the clear. This allowed us to identify the protocols of interest for our analysis, but also revealed the device naming habits of many users as described before. If some sort of social relation between users could be identified (e.g. by a Dropbox community or—as in this case—if they are connected to the same access point), the identification of only one member of this group was instrumental for the identification of multiple other members. In other words, by just identifying a single node in the graph, it was possible to identify nodes for which the hostname information alone was not enough to identify the node uniquely. This happened in every control experiment we performed.

To verify that we in fact did identify the correct course of studies and students with the help of the Dropbox data we made an additional set of control experiments.

We visited two lectures, presented our work and showed a community graph we created from the measurement data which we believed contained multiple students in those lectures. To ensure the privacy of each student, we did not include any potentially sensible data. To verify the graph we asked them (on a voluntary basis) to write down the MAC address of their devices, the hostname, and either their initials, full name or only the first name. For community 54, eight students said to use the wireless campus network and the Dropbox desktop application. They also told us they have a Dropbox share together for their semester. Only four of them had their notebooks with them, we could match all of them to the hashed MAC addresses and names we found for their respective community graph. The first names of the students without a notebook present also matched the names we found. We repeated this for another community with similar results.

Additionally, we searched for publicly accessible social network profiles of the voluntary identified users. Some of the profiles we found had their list of friends visible. We could see the majority of the other already identified group members as their friends. After hashing the names in the list of friends we could also identify other students from the community which we previously could not identify because the LDAP search showed multiple students for the found names. An overview for all analyzed communities can be seen in table 7.1.

	Members	Hostnames	Identified
Community 19	19	15	10
Community 39	42	27	8
Community 50	14	14	2
Community 54	16	14	8

Table 7.1: Results for the Dropbox communities

7.4.6 Countermeasures

Broadcast/multicast protocols fulfill important tasks such as auto-configuration or service/peer discovery. Switching these services off would clearly not be a satisfying countermeasure to the potential privacy threads detailed above. Often there is no easy technical fix for the protocols themselves as the problem really lies in the ability to combine multiple broadcast sources. While a technical solution is difficult

to achieve, there are steps to protect oneself against this kind of analysis which are simple, straightforward and fairly obvious.

In particular, the hostname of a device should never contain any kind of name or personal information which can identify a person—that includes initials, nicknames and IDs. As shown, even the first name alone might be enough to identify a user if cross-referenced with other data sources. Unfortunately, as our data has shown, this is a very common practice. Operating system vendors can actually help advising the user to create better device and service names as many of them make an initial suggestion for a device name. Most users are unaware of the fact that these names will be seen in the clear on the network. Users have to consider the tradeoff between having a unique name as hostname which for example helps to identify a device when using network shares, and a common or random hostname which helps to protect the privacy at the cost of convenience.

Secondly, restricting the data visible to non-friends on a social network profile is helpful if one does not want outsiders to recreate a social graph. This is obviously true for the social network alone, but combining this data with trivially accessible information on the network reveals much more personal information.

A third countermeasure could be the deactivation of the Dropbox LAN sync protocol in the Dropbox desktop application (or any other broadcast-utilizing protocol or application), but we do not recommend this as a general solution. This protocol helps reducing the traffic volume if a share can be found in the local network. But it is important to keep in mind that leaving this option activated might reveal connections to the persons a user shares folders with.

Besides users, network operators should clearly not have directory services such as LDAP broadly accessible. In addition, broadcasts can be easily controlled (blocked) on the air interface on modern access points which would make this kind of analysis harder (as has been done on the IETF network). The same does not apply to multicast, in particular in IPv6-enabled networks.

Finally, broadcast protocol designers should make sure that as little information as possible can be gleaned from the broadcast messages. If temporary IDs or a control back-channel over a server can be used to achieve the same goal these options should be taken into consideration. We have written an Internet draft [WFW16] summarizing our main findings. This document was adopted by the Internet Area Working Group. Together with the IETF we attempt to detail a set of considerations for broadcast protocol designers, which include e.g. the ability to control broadcasts

features on e.g. an SSID level, tighter control of message frequencies, advice on the use of persistent identifiers and others.

7.5 Anomalies

We were presented with several challenges when analyzing the data. Our goal was to automate processes as much as possible since the data set was large and we wanted to build software that eventually runs on live data streams instead of pre-recorded traffic traces. We were surprised to see a large number of anomalies, that made automatic processing challenging.

7.5.1 Number of MAC Addresses

We found 35,386 unique MAC addresses in the traffic records. To our surprise, 16,358 of these addresses had an unknown vendor, from these 2,980 were multi- or broadcast addresses. After an analysis of the packets of those, we found out that these unknown addresses are most likely a failed experiment from some students. Most packets contained a UPnP payload which overwrote the fields of the Ethernet header, causing many unknown vendors for the MAC addresses. To filter those packets, we only counted MAC addresses which had sent at least one valid IP, IPv6, ARP or RARP packet. This resulted in 21,179 unique MAC addresses. The top ten vendors (matched with the OUI list[2] from the IEEE from 30. March 2015) for those can be found in table 7.2.

After we had gathered the announced hostnames for each MAC address as described before, we realized that some hostnames showed up for multiple MAC addresses, even though the hostnames had some unique characteristics like the full name of the user. Most of those multiple MAC addresses also had an unknown vendor, which made us suspicious. It turned out that the broadcast traffic of the university's VPN was visible on the wireless network. The OpenVPN clients generate random MAC addresses which were visible in our traces, which explained the high number of unknown vendors and therefore polluted our data set.

[2] http://standards-oui.ieee.org/oui.txt

Vendor	#
Apple	5438
unknown	5259
Samsung	3059
Intel Corporation	1610
Hon Hai Precision Ind. Co., Ltd.	1101
HTC Corporation	726
Murata Manufacturing Co., Ltd.	678
Sony	575
Liteon Technology Corporation	543
LG Electronics	475
Sum	19464
Others	1715

Table 7.2: Top ten vendors by the number of MAC addresses

7.5.2 Multiple Hostnames per MAC Address

While analyzing the hostnames, we found that some devices broadcasted multiple hostnames. One obvious cause was when a connecting device believed its hostname was already taken, it added a number to its hostname. In other cases, devices acted as a sleep proxy for inactive devices, answering on their behalf [CK13]. It is also possible that users changed their hostnames themselves, reinstalled the operating system or sold their device.

7.5.3 IGMP

Overall the IGMP traffic was very low, but while generating some figures we saw that this traffic increased by more than a factor of ten after the 11^{th} of March, 2015 from 31 kB per day to 376 kB per day. Before that date, only one device announced a multicast group multiple times per day. On that day another device began broadcasting a multicast group once per second. This lead to times at night when this traffic accounted for 37.4% of the overall hourly traffic, even though this is still in the range of kilobytes. Given that broadcast volume is relative small, minor changes, misconfigurations and actions by single hosts can profoundly impact

statistics.

7.6 Future Work

For this initial analysis of the gathered data, we only took a few protocols into account. There are still some other applications which we believe send interesting broadcast data we can use to identify more users and learn more about them. We want to take a look at the Simple Service Discovery Protocol which announces UPnP services and may also contain personal information. We also saw multicast DNS packets indicating that the devices had the Google Chromecast plugin installed. This could help separating desktop devices from mobile- and server-devices. Also, some installations of Microsoft Office for Mac sent out data, which can help with e.g. device or OS fingerprinting.

Due to an overall low traffic volume in our network, we did not investigate the broadcast packets sent by Spotify, Steam and BitTorrent Sync, but we could see that those also include unique IDs and data which might be worth exploring further in our future work.

In this chapter, we only considered data seen by a passive observer. With the data at hand it might be possible to forge answers to the broadcasts sent by individual devices to gather even more information. A first target for this kind of analysis could be devices announcing UPnP services to find details about the media they offer.

Another experiment we want to conduct is to do a replay attack for certain protocols in a controlled environment. This might make it possible to fake the identity of another device or disturb the services from specific applications.

7.7 Related Work

There is an existing body of work that has investigated the properties of campus network traffic and the Dropbox protocol.

The paper from Singh et al. analyzes broadcast data from a campus network to classify packets as normal and anomalous traffic [RS12]. This work examines—like ours—only broadcast data, but considers only the IP addresses of a sender to classify the sender as genuine, malicious or unidentified. The content of the broadcast packets is not further analyzed and no cross-protocol analysis is done. Kotz/Essien analyze the usage patterns in a large campus wireless network by observing the

complete traffic passively for eleven weeks [KE05]. They found that "residential traffic dominated all other traffic". In contrast to our campus, the wireless network included residential buildings, therefore we can expect a different usage pattern in our data. However, a follow-up paper by Henderson et al. analyzing the same campus-wide network shows that the usage patterns changed dramatically from web traffic to peer-to-peer and streaming traffic [HKA04]. We did not consider usage patterns as part of our analysis as we only used broadcast data. Balachandra et al. did a similar analysis of user behavior and network performance but for a public wireless LAN [BVBR02]. Lee et al. analyzed the TCP/IP traffic for a campus wireless network with a focus "on the interarrival times of TCP flows and the number of packets within a flow" [LF09].

Given the popularity of Dropbox, it is not surprising that it has been analyzed before. Kholia/Wegrzyn e.g. analyzed the security aspects of Dropbox by reverse engineering frozen Python applications [KW13]. They describe how the *host_int* is received by the Dropbox application from the Dropbox server on startup and that it can—like we have done—be obtained from the Dropbox LAN sync protocol. Drago et al. characterize Dropbox by a passive measurement of all Dropbox protocol exchanges, i.e. not only broadcasts, to quantify its impact on the network [DMMM+12]. This work does not explicitly address privacy issues but describes that Dropbox announces unique identifiers for the user (*host_int*) and its shares (*namespaces*).

Generally speaking, using non-broadcast traffic requires the observer to be on-path, which is not possible in the general case. We only used data that any device on the network can trivially access without being in a special location or having to rely on certain privileges.

For a lot of the protocol information disclosed by various protocols, the privacy implications have been understood and work is in progress to specify mechanisms against the misuse of this information. E.g. MAC address randomization has been worked on for some time. iOS e.g. does Wifi scans with randomized MAC addresses [iOS15]. Also, the IEEE 802 EC Privacy Recommendation Study Group has looked into MAC address randomization but no standard has emerged as of now.

There is also ongoing work in the IETF that relates to our analysis. E.g. the current host naming practices is being discussed in [HT15] and hostname randomization advice is given. A related document discusses issues found with DHCP configuration information [HMK15].

The problem with all of the above is that point solutions that fix one protocol

are not enough as we have demonstrated in this chapter. E.g. if MAC addresses are frequently changed, the *host_int* of the Dropbox client will render this effort useless as a persistent application layer protocol identifier is broadcast frequently. Therefore, not only traditional network protocol engineers need to work on solutions to these problems but also application developers need to be aware of the privacy implications described above. This was our main motivation to write the Internet draft [WFW16] mentioned before.

7.8 Conclusions

An increasing number of protocols make use of broadcasts for various purposes. These protocols analyzed in isolation do usually not reveal much useful information that would give away personal information of a network user, i.e. do not reveal the real identity of the user. This can change when combing information from multiple broadcast protocols and publicly available data like we have done for this work to gather as much data as possible about the network's user base, i.e. personal information. Even though initial identification of some users came directly from carelessly set hostnames, others could be identified when looking at the social graph the Dropbox LAN sync protocol reveals. We were surprised by how much data we could collect and how easy it was to identify a large number of users of the network as well as their social contacts. This is indeed troublesome, since broadcasts are trivially accessible by anyone in the broadcast domain. Furthermore, there are protocols that include IDs with which a user can be tracked quite precisely since the ID does not change often and the protocol broadcasts that ID with a high frequency. "Traditionally" such IDs were e.g. the MAC address of an interface, but it turns out that the MAC address alone today can be misleading sometimes since e.g. sleep proxies respond to queries on the behalf of other devices. With multiple unique IDs to choose from, tracking users becomes increasingly simple and once the connection between any of these IDs to real users is made, it becomes more and more easy to identify others using these as a starting point.

The protocols in use are only partly to blame. Their design can certainly be improved in ways to better hide information, but the biggest problem is the use of names or other identifiers in hostnames which is a common practice for a large fraction of users.

In the particular network we analyzed, we had additional "support" of an LDAP

server, which certainly helped in automating the identification of users, their current semester (using a heuristic) and course of study. With some extra work, this information (and sometimes much more) was often accessible through other means such as social networks. It remains future work to apply automation using e.g. Google's people search API or the Twitter API for this process. The other network we analyzed had no LDAP server we could use for this purpose but a public meeting attendees list was available. Generally speaking, this helped us to analyze the network's user base, but a real attacker likely does not even need this information. In particular if the attacker tries to attack a certain owner's devices, the victim is already known and the challenge is to figure out which device is owned by the victim. Broadcast data, as shown in this chapter, can make this a trivial undergoing.

On a final note, the analysis presented in this work is still ongoing. Not all protocols have been included and the degree of automation can still be improved to being able to work on live data streams. Also, automatically including external data sources using the APIs mentioned before remains part of our ongoing efforts, which is difficult though as anonymized data cannot be used.

Acknowledgment

This work has received funding from the European Union under the FP7 Grant Agreement n. 318627, project "mPlane".

We would like to thank the many students that have volunteered to take part in the various control experiments as well as the NOC staff of the IETF Meeting in Prague and Yokohama. We are also grateful for the support of the IETF chair Jari Arkko and the various IETF participants that helped sorting out the legal questions of our experiments. Finally, we would like to acknowledge the invaluable support of the DFN legal department, in particular Hagen Sporleder.

8 Conclusions and Outlook

In this dissertation, platforms for active measurements on the Internet have been researched. A number of projects dealing with such platforms with a variety of offered measurement methods already exist. These platforms differ in a number of aspects, e.g. whether they are based solely on software or are pre-installed on special hardware. Some platforms deploy probes on end-devices whereas other target home gateways. They also differ in the degree of freedom they offer, meaning how much an actual user of the platform can do. Some allow for users to schedule their own measurements, other projects are built as research platforms where only experimenters can execute measurements. The presented measurement platform GLIMPSE tries to be both: an application for the end-user to conduct measurements from and in their home network, and a platform for researchers to execute large-scale measurements with a great number of distributed probes.

Multiple measurement campaigns were conducted using GLIMPSE answering a number of research questions and showing the potential power of such a platform. For this work, another focus was to develop the platform, test GLIMPSE as a whole and its individual measurements, and to evaluate potential extensions of the probes (passive measurements). The results of these campaigns showed the accuracy of some of the implemented measurements when compared to existing, well-tested and ubiquitously deployed and used tools. Also, multiple anomalies on the Internet could be observed with the help of GLIMPSE. E.g. a comparison between the Flash-based measurement of Speedtest.net and the HTTP download measurement of GLIMPSE showed that the quality and trust-ability of the results greatly depend on the selected server to download from. Compiled and locally deployed measurement applications like GLIMPSE can implement more complex measurements while providing additional information which a website with a Flash plugin just can not deliver for technical reasons.

In another case, an ISP was found to most likely prioritize measurement traffic to Speedtest.net. A larger deployment of GLIMPSE could be used to investigate further

if multiple ISPs do this. Some of the other anomalies found can only be detected when running long-term measurement campaigns, like the described performance changes to a cloud hosting servers.

The research on broadcast data in this dissertation showed the need for guidance on broadcast protocol design in the context of privacy. Therefore, an Internet Draft has been written which lists considerations for protocol designers which were formulated based on observed protocol behaviour and respective threats to privacy. This document has been adopted by the Internet Area Working Group of the IETF and is actively progressed.

There are some ongoing research projects concerning themselves with network measurements. The Measuring Mobile Broadband Networks in Europe (MONROE) project e.g. that started in mid 2016 attempts to design and implement a platform for large-scale, multi-homed measurements of performance of mobile broadband networks [MON]. Multiple involved partners are going to contribute measurement methods and design measurement campaigns on this platform. The project "Mapping of Broadband Services in Europe" is planed to go online in 2017 and will aggregate measurement results for online connectivity in a central database and interactive application [TÜ].

These two examples show that it is still necessary to perform research in this area and that platforms for integration and aggregation of different measurement methods and results are the next step towards a global monitoring system for things such as networking anomalies.

Bibliography

[AFT11] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring multipath routing in the internet. *IEEE/ACM Transactions on Networking*, 19(3):830–840, 2011.

[AP07] Mark Allman and Vern Paxson. Issues and etiquette concerning use of shared measurement data. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 135–140, New York, NY, USA, 2007. ACM.

[APB09] M. Allman, V. Paxson, and E. Blanton. Tcp congestion control. RFC 5681, RFC Editor, September 2009. `http://www.rfc-editor.org/rfc/rfc5681.txt`.

[ARD] ARD/ZDF-Onlinestudie 2015. `http://www.ard-zdf-onlinestudie.de/index.php?id=530`. Accessed: 2016-05-13.

[att15a] multicast/broadcast experiment at ietf94 (email thread), November 2015.

[att15b] Network experiment during the meeting (email thread), July 2015.

[att15c] No network experiment during the meeting (email thread), July 2015.

[BEBS16] Trevor Burbridge, Philip Eardley, Marcelo Bagnulo, and Juergen Schoenwaelder. Information model for large-scale measurement platforms (lmap). Internet-Draft draft-ietf-lmap-information-model-10, IETF Secretariat, July 2016. `http://www.ietf.org/internet-drafts/draft-ietf-lmap-information-model-10.txt`.

[BGLM08] V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E.L.J.S. Mech. Fast unfolding of communities in large networks. *J. Stat. Mech*, 2008.

[Bra96] Scott O. Bradner. The internet standards process – revision 3. BCP 9, RFC Editor, October 1996. http://www.rfc-editor.org/rfc/rfc2026.txt.

[Bri] Brian Trammell. mPlane Architecture and Protocol. https://www.ict-mplane.eu/sites/default/files//public/public-page/mplane-workshop-platform-measuring-internet//1309mplanearchfinal.pdf. Accessed: 2016-06-08.

[Bro15] Broadband Forum. Broadband Access Service Attributes and Performance Metrics. (TR-304), 02 2015.

[Bun] Statistisches Bundesamt. Ausstattung privater Haushalte mit Informations- und Kommunikationstechnik - Deutschland. https://www.destatis.de/DE/ZahlenFakten/GesellschaftStaat/EinkommenKonsumLebensbedingungen/AusstattungGebrauchsguetern/Tabellen/Infotechnik_D.html. Accessed: 2016-05-13.

[BVBR02] Anand Balachandran, Geoffrey M. Voelker, Paramvir Bahl, and P. Venkat Rangan. Characterizing user behavior and network performance in a public wireless lan. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '02, pages 195–205, New York, NY, USA, 2002. ACM. http://doi.acm.org/10.1145/511334.511359.

[CBS12] Igor Canadi, Paul Barford, and Joel Sommers. Revisiting broadband performance. *Proceedings of the 2012 ACM conference on Internet measurement conference - IMC '12*, page 273, 2012.

[cHK+09] k. claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov. Internet Mapping: from Art to Science. In *IEEE DHS Cybersecurity Applications and Technologies Conference for Homeland Security (CATCH)*, pages 205–211, Watham, MA, Mar 2009.

[CK13] S. Cheshire and M. Krochmal. Multicast dns. RFC 6762, RFC Editor, February 2013. http://www.rfc-editor.org/rfc/rfc6762.txt.

[Dis10] Glasnost: Enabling end users to detect traffic differentiation. *Proceedings of the 7th ...*, pages 27–27, 2010.

[DMMM+12] Idilio Drago, Marco Mellia, Maurizio M. Munafo, Anna Sperotto, Ramin Sadre, and Aiko Pras. Inside dropbox: Understanding personal cloud storage services. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, pages 481–494, New York, NY, USA, 2012. ACM.

[EEBW11] P. Eardley, L. Eggert, M. Bagnulo, and R. Winter. How to contribute research results to internet standardization. RFC 6417, RFC Editor, November 2011. `http://www.rfc-editor.org/rfc/rfc6417.txt`.

[eMa] eMarketer Inc.). 2 Billion Consumers Worldwide to Get Smart(phones) by 2016. `http://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694`. Accessed: 2016-06-02.

[EMB+15] P. Eardley, A. Morton, M. Bagnulo, T. Burbridge, P. Aitken, and A. Akhter. A framework for large-scale measurement of broadband performance (lmap). RFC 7594, RFC Editor, September 2015.

[FW15] Michael Faath and Rolf Winter. Measurements with the Masses. In *IRTF & ISOC Research and Applications of Internet Measurements (RAIM) Workshop*, Yokohama, Japan, 2015.

[FW16] Michael Faath and Rolf Winter. A GLIMPSE of the Internet's Fabric. In *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 49–54. ICST, 2016.

[FWW15] M. Faath, R. Winter, and F. Weisshaar. A cautious look at using Internet standards-to-be in research work. In *IEEE Conference on Standards for Communications and Networking (CSCN), 2015*, pages 13–17, Oct 2015.

[FWW16] M. Faath, R. Winter, and F. Weisshaar. How broadcast data reveals your identity and social graph. In *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 357–362, Sept 2016.

[GF05] Saikat Guha and Paul Francis. Characterization and measurement of tcp traversal through nats and firewalls. In *Proceedings of the*

5th ACM SIGCOMM Conference on Internet Measurement, IMC '05, pages 18–18, Berkeley, CA, USA, 2005. USENIX Association.

[HKA04] Tristan Henderson, David Kotz, and Ilya Abyzov. The changing usage of a mature campus-wide wireless network. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, MobiCom '04, pages 187–201, New York, NY, USA, 2004. ACM. http://doi.acm.org/10.1145/1023720.1023739.

[HMK15] Christian Huitema, Tomek Mrugalski, and Suresh Krishnan. Anonymity profile for dhcp clients. Internet-Draft draft-ietf-dhc-anonymity-profile-04, IETF Secretariat, October 2015. http://www.ietf.org/internet-drafts/draft-ietf-dhc-anonymity-profile-04.txt.

[HT15] Christian Huitema and Dave Thaler. Current hostname practice considered harmful. Internet-Draft draft-ietf-intarea-hostname-practice-00, IETF Secretariat, October 2015. http://www.ietf.org/internet-drafts/draft-ietf-intarea-hostname-practice-00.txt.

[IETa] IETF Working Group IP Performance Metrics (ippm). Charter for Working Group. https://datatracker.ietf.org/wg/ippm/charter/. Accessed: 2016-05-02.

[IETb] IETF Working Group Large-Scale Measurement of Broadband Performance (lmap). Charter for Working Group. https://datatracker.ietf.org/group/lmap/charter/. Accessed: 2016-05-02.

[iOS15] ios security, September 2015.

[KB13] Kristina Knaving and Staffan Björk. Designing for fun and play: Exploring possibilities in design for gamification. In *Proceedings of the First International Conference on Gameful Design, Research, and Applications*, Gamification '13, pages 131–134, New York, NY, USA, 2013. ACM.

[KE05] David Kotz and Kobby Essien. Analysis of a campus-wide wireless network. *Wirel. Netw.*, 11(1-2):115–133, January 2005. http://dx.doi.org/10.1007/s11276-004-4750-0.

[KW13] Dhiru Kholia and Przemyslaw Wegrzyn. Looking inside the (drop) box. In *Presented as part of the 7th USENIX Workshop on Offensive Technologies*, Berkeley, CA, 2013. USENIX.

[KWNP10] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: Illuminating the edge network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 246–259, New York, NY, USA, 2010. ACM.

[LF09] Ian W. C. Lee and Abraham O. Fapojuwo. Analysis and modeling of a campus wireless network tcp/ip traffic. *Computer Networks*, 53(15):2674–2687, 2009. `http://dblp.uni-trier.de/db/journals/cn/cn53.html#LeeF09`.

[Lina] Linode, LLC. Linode Facilities Speedtest. `https://www.linode.com/speedtest`. Accessed: 2016-06-21.

[Linb] Linode, LLC. Linode Website. `https://www.linode.com/`. Accessed: 2016-06-21.

[LMWC15] Weichao Li, Ricky K P Mok, Daoyuan Wu, and Rocky K C Chang. On the accuracy of smartphone-based mobile network measurement, 2015.

[Luc] Matthew Luckie. Scamper: a Scalable and Extensible Packet Prober for Active Measurement of the Internet.

[Mar] Marco Mellia. Overview of the mPlane project. `https://www.ict-mplane.eu/sites/default/files//public/public-page/mplane-workshop-platform-measuring-internet//1317mplanefinalworkshopheidelberg.pdf`. Accessed: 2016-06-07.

[Mat] Matthias Lee. pytesseract 0.1. `https://pypi.python.org/pypi/pytesseract/0.1`. Accessed: 2016-06-12.

[Mog84] Jeffrey Mogul. Broadcasting internet datagrams. STD 5, RFC Editor, October 1984. `http://www.rfc-editor.org/rfc/rfc919.txt`.

[Moh15] Mohamed Ahmed. WP7 – Dissemination, Exploitation and Standardization. Summary at Y3. mPlane Consoritum, 2015.

[MON] MONROE. MONROE Website. https://www.monroe-project.eu. Accessed: 2016-07-10.

[mPla] mPlane Consortium. mPlane deliverables. https://www.ict-mplane.eu/public/public-deliverables. Accessed: 2016-06-16.

[mPlb] mPlane Consortium. mPlane registry. http://www.ict-mplane.eu/registry/core. Accessed: 2016-05-18.

[mPlc] mPlane consortium. mPlane website. https://www.ict-mplane.eu/. Accessed: 2016-06-15.

[ngi] nginx. nginx Website. https://nginx.org/. Accessed: 2016-06-18.

[Ooka] Ookla. How Speedtest.net Works - How does the Begin Test button select a server? https://support.speedtest.net/hc/en-us/articles/203845410-How-does-the-Begin-Test-button-select-a-server-. Accessed: 2016-06-12.

[Ookb] Ookla. Speedtest.net website. http://www.speedtest.net/. Accessed: 2016-06-15.

[Ope] OpenWrt developer team. OpenWrt website. https://www.openwrt.org/. Accessed: 2016-08-01.

[PDMC03] R Prasad, C Dovrolis, M Murray, and KC Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *Network, IEEE*, 2003.

[Pos81] J. Postel. Internet control message protocol. STD 5, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc792.txt.

[RIP] RIPE NCC. RIPE Atlas website. https://atlas.ripe.net. Accessed: 2016-06-15.

[RS12] R.K. Singla Raman Singh, Harish Kumar. Traffic analysis of campus network for classification of broadcast data. *International Conference on Intelligent Infrastructure*, 2012.

[RTMM14] Arianna Rufini, Edion Tego, Marco Mellia, and Francesco Matera. Multilevel bandwidth measurements and capacity exploitation in gigabit passive optical networks. 2014.

[Sam] SamKnows. SamKnows website. `https://www.samknows.org/`. Accessed: 2016-08-01.

[Sam15] SamKnows. Samknows Test Methodology. Technical report, 02 2015. `https://www.samknows.com/broadband/uploads/methodology/SQ301-005-EN-Test-Suite-Whitepaper-4.pdf`.

[SB12] Joel Sommers and Paul Barford. Cell vs. wifi: On the performance of metro area mobile connections. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, pages 301–314, New York, NY, USA, 2012. ACM.

[SB16] Juergen Schoenwaelder and Vaibhav Bajpai. A yang data model for lmap measurement agents. Internet-Draft draft-ietf-lmap-yang-05, IETF Secretariat, July 2016. `http://www.ietf.org/internet-drafts/draft-ietf-lmap-yang-05.txt`.

[SBF14] Srikanth Sundaresan, Sam Burnett, and Nick Feamster. BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks. *Usenix ATC*, 2014.

[Sik] Sikuli developers. Sikuli website. `http://www.sikuli.org/`. Accessed: 2016-06-12.

[SOB+13] Mario A. Sánchez, John S. Otto, Zachary S. Bischof, David R. Choffnes, Fabián E. Bustamante, Balachander Krishnamurthy, and Walter Willinger. Dasu: Pushing experiments to the internet's edge. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 487–499, Lombard, IL, 2013. USENIX.

[SOB+14] M.A. Sanchez, J.S. Otto, Z.S. Bischof, D.R. Choffnes, F.E. Bustamante, B. Krishnamurthy, and W. Willinger. A measurement experimentation platform at the internet's edge. *Networking, IEEE/ACM Transactions on*, PP(99):1–1, 2014.

[Son13] Netradar – Measuring the Wireless World. In *Wireless Network Measurements*, pages 29–34, 2013.

[Spo15] H. Sporleder. Dein Name ist Programm. *DFN Infobrief Recht*, pages 16–18, November 2015.

[Tel] Deutsche Telekom. Leistungsbeschreibung MagentaZuhause. `http://www.telekom.de/dlp/agb/pdf/43840.pdf`. Accessed: 2016-06-01.

[The] The Qt Company. Qt website. `https://www.qt.io/`. Accessed: 2016-07-02.

[TMF+15a] Brian Trammell, Marco Mellia, Alessandro Finamore, Stefano Traverso, Tivadar Szemethy, Balazs Szabo, D Rossi, Benoit Donnet, Fabrizio Invernizzi, and Dimitri Papadimitriou. Final implementation of software and libraries - final release. (D2.3), 06 2015.

[TMF+15b] Brian Trammell, Marco Mellia, Alessandro Finamore, Stefano Traverso, Tivadar Szemethy, Balazs Szabo, D Rossi, Benoit Donnet, Fabrizio Invernizzi, and Dimitri Papadimitriou. mplane architecture specification. (D1.4), 04 2015.

[TÜ] TÜV Rheinland Consulting. Mapping of Broadband Services in Europe Website. `https://www.broadbandmapping.eu/`. Accessed: 2016-07-05.

[Uni] University of California. BOINC Website. `https://boinc.berkeley.edu/`. Accessed: 2016-07-05.

[WFW16] Rolf Winter, Michael Faath, and Fabian Weisshaar. Privacy considerations for ip broadcast and multicast protocol designers. Internet-draft, IETF Secretariat, October 2016. `https://www.ietf.org/internet-drafts/draft-intarea-broadcast-consider-01.txt`.

[Wir] Wireshark. Wireshark website. `https://www.wireshark.org/`. Accessed: 2016-06-05.

Zusammenfassung

In dieser Dissertation wurden Plattformen für aktive Messungen im Internet untersucht. Eine Beschreibung bereits vorhandener Forschungsarbeiten und insbesondere Bemühungen zur Standardisierung solcher Plattformen geben einen Überblick über den aktuellen Stand der Forschung. Durch eine Zusammenfassung der aktuell verfügbaren Plattformen zeigt sich, dass es diese sich in vielen Aspekten unterscheiden, wie etwa die Art der Implementierung (als Software-Pakete oder vorinstalliert auf Hardware), dem Grad der Freiheit für den Benutzer (können Benutzer oder nur die Plattform-Betreiber beziehungsweise Forscher Messungen durchführen) und die verwendeten Messmethoden und deren Erweiterbarkeit.

Basierend auf den Arbeiten des EU-geförderten Forschungsprojektes mPlane und den vorläufigen Standards der Large-Scale Measurement of Broadband Performance (LMAP) Arbeitsgruppe der Internet Engineering Task Force (IETF), wurde für diese Dissertation GLIMPSE (Global Internet Monitoring and Probing System) entwickelt. Diese Plattform für aktive Messungen ist speziell für den Einsatz bei Endnutzern im Heimnetzwerk konzipiert und entwickelt worden. Neben einer detaillierten Beschreibung des Systems und der implementierten Messmethoden sind außerdem die Ergebnisse mehrerer Messkampagnen dargestellt. Es zeigt sich zum Einen, welche Art von Messungen durchgeführt und welche Anomalien dabei entdeckt werden können. Zum Anderen wird aber auch deutlich, dass Messkampagnen langfristig und auf einer großen Anzahl von verteilten Punkten im Internet durchgeführt werden müssen, um Unregelmäßigkeit frühzeitig zu erkennen.

GLIMPSE befasst sich ausschließlich mit aktiven Messungen. Um das Forschungspotenzial von passiven Messungen auf Endgeräten zu demonstrieren, wurde eine Analyse der passiv empfangenen Broadcast- und Multicastdaten auf solchen Geräten durchgeführt. Die Ergebnisse dieser Forschung wurde in die Internet Area Arbeitsgruppe der IETF eingebracht und als Dokument der Arbeitsgruppe angenommen. Dieser sogenannte Internet Draft gibt Hinweise und Beispiele für Entwickler, die Broadcast- und Multicastprotokolle designen und hat das Ziel, zum RFC zu werden.