FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Monitoring Net Neutrality of Austrian ISPs

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Thomas Schreiber
Matrikelnummer 1054647

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl

Wien, 10. Oktober 2016

Thomas Schreiber	Edgar Weippl

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Monitoring Net Neutrality of Austrian ISPs

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Thomas Schreiber
Registration Number 1054647

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl

Vienna, 10th October, 2016

Thomas Schreiber	Edgar Weippl

Erklärung zur Verfassung der Arbeit

Thomas Schreiber
Liechtensteinstraße
2170 Poysdorf

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

Wien, 10. Oktober 2016

Thomas Schreiber

Kurzfassung

Netzneutralität ist ein Terminus, der in Wirtschaft, Politik und Technik stark diskutiert wird. Im Grunde bedeutet Netzneutralität, dass alle Datenströme im Internet, unabhängig ihres Inhalts, gleich behandelt werden müssen. Praktisch bedeutet das, dass es Internetprovidern (ISPs) nicht erlaubt ist, ihre eigenen Webservices oder die Services von Partnern anderen gegenüber zu bevorzugen. Eine solche bevorzugte Behandlung könnte etwa eine erhöhte Geschwindigkeit oder Verfügbarkeit sein, andererseits aber auch in einem unterschiedlichen Verrechnungsmodell liegen.

In der vorliegenden Diplomarbeit wird nicht speziell auf wirtschaftliche oder politische Motive und Verstöße eingegangen, sondern Netzneutralität aus einem rein technischen Standpunkt betrachtet.

Dazu wurde ein eigenes Messsystem entwickelt. So wurden fünf Anschlüsse bei vier österreichischen ISPs angemietet und, um konsistente Aussagen zu ermöglichen, über einen Zeitraum von zwei Monaten über 100.000 Messungen durchgeführt.

Die Messungen belegten beispielsweise den Einsatz eines transparenten Web-Proxies bei T-Mobile Österreich und die Verfälschung von DNS-Abfragen nichtexistierender Domains durch Hutchison Drei.

Abstract

Net neutrality is a topic that is broadly discussed in economy, politics and technology. Basically, net neutrality states, that all internet packets should be treated equally by Internet Service Providers (ISPs). This principle does, for example, not allow ISPs to treat their own web services or the services of partners better than all other web services. The treatment has to be equal in every aspect - be it connection speed, availability or billing.

In this thesis, neither economic nor political aspects of net neutrality are discussed, but the focus is only on technology.

For this, a measurement system was built and five internet plans with four Austrian ISPs were ordered. Over a period of two months, over 100,000 single measurement points were taken.

Evaluating these measurements showed, for example, that T-Mobile Austria uses a transparent web proxy for HTTP connections and that Hutchison Drei tampers with DNS requests for non-existing domains.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Net Neutrality is a topic that is discussed broadly in technical, economic and legal literature[23, 14, 27, 7]. The term "Net Neutrality" itself means that an Internet Service Provider (ISP) should only take on the role of a carrier for packets on the network. The ISP should therefore not treat packets differently based on the protocol, destination, source or other characteristics[5]. Since net neutrality covers a broad spectrum of topics, not every aspect of net neutrality is covered by this thesis. For example, economic aspects like zero-rating are not given any attention.

However, it should be mentioned, that net neutrality has a legal basis in the European Union. This legal framework is found in Regulation (EU) 2015/2120, where recital 11 specifically mentions, that blocking, slowing down, altering, interfering with and discriminating specific content or services that goes beyond reasonable traffic management measures, is prohibited with certain exceptions. This regulatory action is necessary since tampering with net neutrality can have significant economic benefits for ISPs, for example by blocking VoIP traffic to force users to use conventional – billed – methods for communication. Another lucrative use-case would be to force content providers which cause high traffic volumes to pay for fast delivery to the end customer. This was already seen with the American ISP *Comcast*, which throttled traffic with the streaming provider *Netflix*, forcing the company to pay a fee to the ISP for continuing full-speed streaming to end customers[15].

For definitions of what measures affecting net neutrality are allowed within EU regulations, *BEREC*, a consortium consisting of national telecommunication authorities, has published guidelines[27] for providers. Before these guidelines were published, public discussion and concern for net neutrality formed. At the center of this was the collective *savetheinternet.eu*, which organized protests, petitions and reached out to citizens, asking them to contact regulation authorities and politicians.

The BEREC guidelines e.g. define in which cases traffic management and zero rating are allowed. They also explicitly state, that neither blocking, slowing down, restricting or altering traffic constitute allowed traffic management measures. In contrast, ISPs can

provide “specialized” services which they are allowed to optimize, e.g. Voice over LTE. However, they have to be transparent to the customer and provide information about all measures concerning network neutrality.

In Austria, some public discussion spurred when Hutchison Drei announced its *Spotify* packet, allowing customers to stream unlimited music from Spotify without being charged[32]. This zero-rating also has technical violations against net neutrality - when the included data volume is spent, ISPs throttle the bandwidth for customers. If the bandwidth is kept up for some arbitrary services like Spotify, this would constitute a violation[31].

To our knowledge, there is not yet scientific work concerning the technical aspects of net neutrality for Austria. Therefore, in the following chapters, a monitoring system is designed and applied to detect net neutrality violations with Austrian ISPs.

The remainder of this thesis is structured as follows: In chapter 2, the scientific work of other researchers is discussed. These researchers mostly focused on markets other than Austria, but their findings are still relevant to this thesis.

In chapter 3, a measurement system is presented. For this thesis, internet plans with four Austrian ISPs were ordered. All tests were conducted using these four ISPs. For a better consistency, the measurements were done over a period of two months.

These measurements resulted in over 100,000 single measurement points which are evaluated in detail in chapter 4.

Following this, possible background reasons for these results are discussed in chapter 5. Some plans for future measurements are then given in chapter 6. Finally, the most important results are once again summarized in chapter 7.

CHAPTER 2

Related Work

Aside from legal and regulatory documents, many researchers conducted experiments concerning the technical restrictions of network neutrality. Some researchers focus on the part of net neutrality, that states, that ISPs should not manipulate the content of network packets.

For example, *Nakibly et al.* [26] showed that some network operators are manipulating HTTP traffic. In their paper, they showed that not only edge ISPs – as measured in this thesis – but also some intermediate network operators manipulate traffic and therefore affect customers of all edge ISPs in the same way. As a method for content manipulation, these network operators use *out-of-band injection*. This means, that the original packets are not discarded and replaced by a forged packet, but that a forged packet is inserted into the network stream between server and client and poses for the original packet. If the forged packet then arrives first at the client, the client discards the original packet. This method has the advantage that it does not introduce a point of failure – if the forged packet arrives second or contains any errors, the client simply treats the original packet as intended and does not experience any difficulties when retrieving the requested HTTP resources. For their research, Nakibly et al. intercepted all traffic from three universities and one company for multiple weeks, recording over 1.500 terabytes of IP packets with *netsniff-ng* and using a filter to only capture HTTP traffic. They then analyzed all packets and stored all TCP segments that carried different payloads but corresponded to the same sequence number. As a limitation of this methodology, in-place-swapping of packets was not detected. As for the result, the researchers showed that in most cases of out-of-band injection, the forged packet wins the race (78% of the time). The affected packets were mostly coming from Chinese websites. In the manipulated HTTP traffic, some JavaScript-Code was inserted that either served advertisements or malware. Also, the results could not be reproduced, as most content manipulation lasted only for a short period of time, often disappearing after a few hours or a few days.

The paper of Nakibly et al. is relevant to this thesis, as it shows that not all content manipulation is caused directly by the ISP, but could also be attributed to core network

operators. Since a server-centric approach is used in this thesis, this sort of content manipulation should not occur, since both clients and server are located in Austria and serve non-public content - therefore content-manipulation caused by Asian network operators should not take place. As for the out-of-band injection: In this thesis, content manipulation is measured over multiple months. So, if a racing condition decides if the original packet or the forged packet is received by the client, this should show in the results.

The same goal for network operators – generating revenue streams by delivering advertisements to the end customers – can also be reached by other measures. In their research based on 259,000 measurements by 193,000 users, *Weaver et al.* [35] showed that some providers use failed DNS lookups for monetization. They showed, that some ISPs don't return the legitimate *NXDOMAIN* error code for non-existing web pages but instead return a custom page to users. Aside from ISPs, third-party DNS providers use DNS redirection to custom error pages as their primary source of income. These pages usually contain some search results, sponsored content, and advertisements based on keywords extracted from the domain that was requested. Depending on the provider of these pages, ISPs either only redirect failed DNS lookups with hostnames beginning with *www.* or all lookups. Also, some providers only redirect *NXDOMAIN* errors, while others also redirect *SERVFAIL* errors. The revenue is substantial: ISPs can make a profit of 1-3 dollars per customer per year by DNS redirection. As a consequence of this research, for this thesis, a DNS lookup test was implemented, where a lookup for non-existing hostnames is performed. Similar to the paper of Weaver et al., the test is only counted as passed, if the original *NXDOMAIN* error code is returned by the ISP, instead of some custom error page.

Another research that deals with content manipulation and blocking was done by *Khattak et al.* [21]. In their paper, they investigated if Tor users were served other HTTP content than non-Tor users. As a methodology, they chose two different methods. First, they decided to scan the entire IPv4 address space for manipulation when connecting with an HTTP request on TCP port 80. As a second measurement, they requested the homepages of the top thousand websites as listed on Alexa. They showed, that around 6.8% of HTTP requests are blocked when using Tor while being served normally without using a Tor connection. The largest provider responsible for these blockings was the content delivery network *CloudFlare*, which accounted for 2.5%. In their paper, they only measured the homepage of each respective site, therefore not taking content manipulation or blocking into account that does not occur on the homepage but only on subpages, e.g. on Wikipedia, where Tor-users may only read, but not edit articles.

For this thesis, this is relevant, as based on this paper, a test based on Tor OONI was introduced, that records any content manipulation that takes place when using Tor. In contrast to the paper of Khattak et al., in this thesis only a few selected websites are accessed using Tor, in this case, websites that are blocked by Austrian providers like *kinox.to*.

Research that is also very relevant for this thesis has been done by *Xu et al.* [37]. In their paper, they discovered that all four mobile ISPs in the United States use proxies for

their networks. Especially, port 80 is handled over a split TCP connection and a proxy. Providers in the U.S. use various techniques to "optimize" traffic but are not transparent to the customer. The researchers discovered at least five different optimizations. First, some ISPs cache web resources like images or stylesheets for a certain period of time. This can be measured, if one has control over both client and server and requests that were sent by the client never reach the server as they are handled directly by the proxy. Another optimization is DNS redirection: If an HTTP request contains a "Host" field, the target IP of the IP packet is ignored by the proxy. Also, content is manipulated: Certain proxies optimize files in the U.S., e.g. whitespace is trimmed or images are transcoded to reduce mobile network traffic. Another optimization is the automatic addition of the "connection: keep-alive"-header to HTTP responses or the dropping of the *FIN* TCP-packet. Lastly, some ISPs delay the TCP handshake between their middleboxes and the server until they get an HTTP request from the client.

These tests all are relevant for this thesis and are therefore conducted for the Austrian ISPs. The tests, however, had to be rewritten, as Xu et al. did not provide any code of their work. Some Austrian ISPs use proxies with techniques similar to the optimizations found in the paper, as is shown in a later chapter of this thesis.

Many researchers focus on censorship of certain protocols and politic contents. For this, the *RIPE Atlas* network has been built, spanning over 13,000 instances worldwide where researchers can conduct measurements. The network can be rented for experiments with *credits*. These credits can be easily generated by lending network clients to the Atlas network for multiple years.

For example, the Atlas network was used to monitor content blocking in Turkey and Russia. *Anderson et al.* [3] showed in their paper how the governments of these countries facilitated the blockings and in which timeframe and density they occurred.

For this thesis, using the RIPE Atlas network for measurements is not possible since the network is limited to only certain tests designed to measure censorship (e.g. ping, traceroute, sslcert, DNS)[4]. Also, the hardware poses certain limitations as different iterations are used for test instances and affect measurements. Hard- and software for the test clients are continually updated, but dealing with different software and hardware revisions inflicts additional work when designing the measurement metrics.

The mentioned papers are however still relevant as they suggest how censorship techniques work in general and the measurements used in this thesis are heavily inspired by the methodology of the RIPE Atlas network. Similar to Atlas, ping, traceroute and DNS tests are conducted to measure censorship and speed of different websites.

In November 2015, shortly before the design of the tests used in this thesis, T-Mobile USA introduced *BingeOn*, a technique that allows American customers to stream videos from participating websites without affecting the data volume included in their mobile plan. Even before, T-Mobile USA provided a similar program for music streaming, *Music Freedom*, which caused controversy as smaller radio stations were either not included in the program or had to wait for long periods to be included[39].

In Austria, Hutchison Drei provides a similar service with *Spotify*, where users can stream an unlimited amount of music via Spotify without affecting their data plan[32].

While this Zero rating is a violation of net neutrality – even if not in a technical sense – T-Mobile USA goes one step further and limits the bandwidth of websites participating in the BingeOn-program. This caused quite a bit of controversy in the U.S., led to the EFF suggesting that T-Mobile abandons the BingeOn-program and various responses by John Legere, the CEO of T-Mobile USA.

Kakhi et al. [19] focused their research on which techniques T-Mobile uses for BingeOn and published a paper discussing their findings. They discovered, that T-Mobile uses policing to reduce bandwidth and is probably using a token bucket system with a very small or no queue which leads to high retransmission rates. Even though T-Mobile claims to "optimize" video, no transcoding is taking place, content, therefore, is not modified. If a user tries to stream eligible video, the bandwidth is automatically limited to 1.5 Mbps. For detecting eligible video streams, T-Mobile seems to be using deep packet inspection, as the selection criteria are matched with a simple string comparison. For HTTP-traffic, depending on the concrete website (e.g. Netflix, HBO Go, Hulu), the *Host*-File of the GET-requests as well as the *Content-Type* is used to categorize traffic. For HTTPS-websites (e.g. YouTube), the server name provided in the Server Name Indication (SNI)-extension during the TLS handshake is used. The researchers showed that by randomizing the fields of the HTTP GET-request header, evading detection by BingeOn is possible. As a consequence, it is possible to zero-rate arbitrary traffic by manipulating these fields. For conducting their measurements, the researchers simply replayed traffic, once over the original channels and once over an encrypted VPN service. This architecture enabled them to compare traffic throughput consistently[20].

This research is directly relevant to this thesis, as Hutchison Drei provides the Spotify Zero-rating mentioned above. For the measurement, a test specifically for detecting video transcoding and rate limiting was designed. Even though this thesis follows a client-server-centric approach, replays of recorded traffic were used for detecting changes in Quality of Experience in RTP traffic used by VoIP.

Tools for detecting net neutrality violations are available in abundance. Many of them are even available as Open Source software. Already mentioned were the RIPE Atlas network that provides measurement clients for detecting censorship and the Tor OONI-framework that detects content manipulation for Tor users. Another tool is *Glasnost*, which is used to detect shaping of traffic using the BitTorrent-protocol. The tool works by actively generating traffic by using a test server and measuring discrimination that is triggered on the TCP layer or on the application layer.

In their research, *Dischinger et al.* [10] found in 2010, based on tests by more than 350,000 users testing over 5,800 ISPs, that traffic shaping for the BitTorrent-protocol or other P2P protocols typically only affects a limited number of users of an ISP. Also, some providers seem to only employ traffic management measures at certain times of a day. For example, Kabel Deutschland discriminated P2P traffic only between 6pm and midnight. The researchers also found, that slowing down P2P traffic may also only affect users with heavy loads or at times with a high load of the network. As a basis for discrimination, most providers inspect the TCP layers and slow down traffic that flows between ports that are associated with certain protocols, e.g. TCP port 6881 for

BitTorrent traffic. Based on these results, this thesis also measures for traffic shaping in regular intervals to be able to detect slowing down only at certain hours of the day.

The tool *NetPolice* tests HTTP, SMTP and VoIP, *Bonfide* tests YouTube-Videos and RTSP traffic by inspecting routing information, packet headers and application layer content. *Netalyzr* detects DNS tampering. Some tools, for example, *NeuBot* or *NANO* work not by generating own test traffic but by passively monitoring user traffic and inspecting it for indications of discrimination.

Not directly affecting network neutrality, but still relevant is the specification of the *RTR Multithreaded Broadband Test* (RMBT)[33]. This protocol defines a speed test technique. Implementations are available for Java and JavaScript using WebSockets. Applications based on the protocol are used by the network regulation authorities in Austria, Slovenia, Serbia and the Czech Republic. In this thesis, all tests that aim to measure network speed are based on a custom python implementation of this protocol.

Even though there are many papers and tools available for detecting net neutrality violations, there is no single tool that meets all requirements for this thesis. These tools either don't cover certain tests, are not possible to run without reporting results to the manufacturer of the tool or do not meet the specific landscape of the test setup. However, most of the implemented metrics and methods are corresponding to published papers of net neutrality violations in other countries. Where possible, third-party tools are used to add reliability and control mechanisms while reducing custom code and possible errors in the implementation. For example, the Tor OONI-measurement framework for detecting HTTP header manipulation is directly used in the test framework of this thesis.

CHAPTER 3

Methodology

In this section, the methodology and scope of the measurements done in this thesis are described. First, the tested providers and used hardware are listed. Following this, the used software architecture and the taken approach when designing the tests is given. Lastly, each measurement is explained in detail, including the motivation and background of the tested technology.

3.1 Tested providers and contracts

All measurements have been done with four Austrian ISPs using two fixed-line products and three mobile (LTE) products. The used products for LTE measurements are:

- A1 Net Cube Internet M ("A1 LTE")
- Hutchison Drei HUI SIM Flat 30 ("Drei")
- T-Mobile My Homenet unlimited light ("TMA")

For the fixed-line clients, the following products were used:

- A1 Internet Pur ("A1 DSL")
- UPC Fiber Power Pack Small ("UPC")

All contracts for the test clients have been concluded by a natural person and not on behalf of any organization or company. So, the products could be obtained by any person and not only by companies. The measurement results, therefore, represent the connection quality obtainable by any customer of these ISPs in the same geographical area.

3.2 Measurement Architecture

3.2.1 Hardware Architecture

For conducting all measurements, a client-server architecture was established. Therefore, a single test server was used. This test server is connected to the Internet via Gigabit LAN, obtaining a static public IPv4 address belonging to the address space of the Vienna University of Technology. This server conducted all measurements with five clients, connected to the respective Austrian ISPs. An overview of this architecture is given in Figure 3.1.

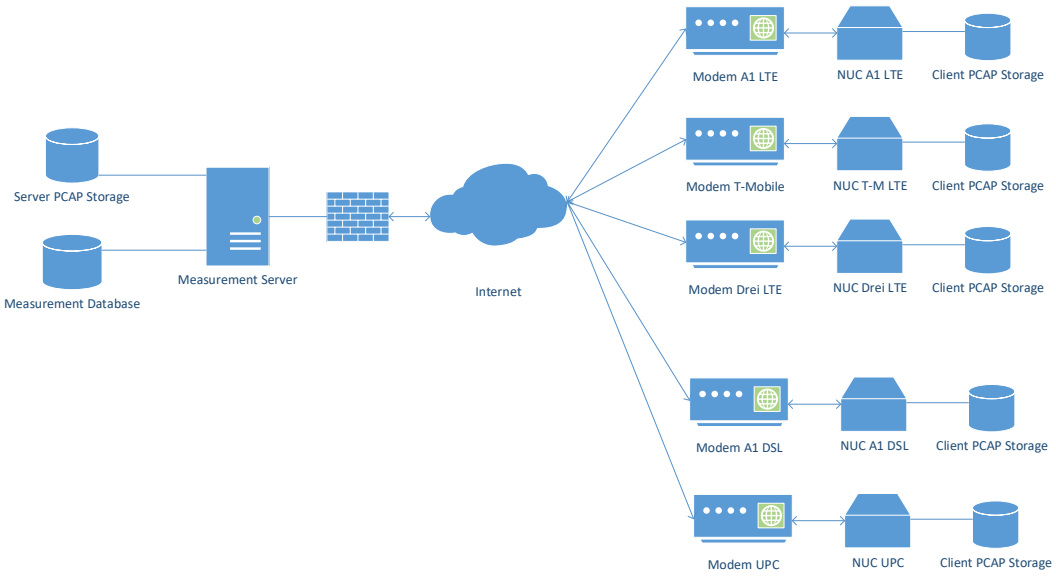

Figure 3.1: Measurement architecture

Since all tests should not be recognizable to the providers during the measurement period, the whole IT landscape was under control of the researchers, the code has not been made public until after all tests have occurred. Also, the server firewall was configured in a way that allowed only connections on port 22 (SSH) from arbitrary IP addresses. So, for all other tests, exceptions for the client IPs had to be made to the firewall. To deal with dynamic IPs, clients first established an SSH connection to the server, wrote their current IP address to a file which in turn was watched for changes by a script on the server. This script then removed any firewall exceptions concerning old IP addresses of the respective client and added a new exception for the given IP, allowing all inbound and outgoing traffic for the address. This also ensured that no foreign traffic would contaminate test results.

When beginning to research, the original plan was to use cheap Raspberry Pi-workstations for test clients. This plan was discarded in favor of more powerful hardware. This upgrade was deemed necessary to guarantee that measurement results are not skewed by hardware limitations.

As a result, for the clients, Intel NUCs were used. These featured an Intel Core i5-6260U CPU clocking at 2x 1.80 GHz, 16 GB of DDR4 RAM and a 500 GB Solid State Drive. Each client is connected to the original router provided by the respective ISP, with the exception of Hutchison Drei, which is connected using a *TP-Link TL-MR6400* modem. So, some test results may be influenced by the used hardware for the router and not the ISP itself. This is discussed later in the test results. We decided to stick to the hardware provided by the ISP to better represent the experience of a typical customer, not a lab environment.

For both clients and server, Ubuntu 16.04 LTS ("Xenial Xerus") was used as an operating system. As can be seen in Figure 3.2, the LTE clients were installed next to each other and placed in next to a window with strong LTE signal. The NUCs with a fixed-line connection were placed in a data center with climate control.

Figure 3.2: Installation of the used NUCs and LTE modems

3.2.2 Software Architecture

This section describes the used software architecture, the requirements that lead to this architecture and the impact of this decisions. For the framework, multiple requirements had to be met:

1. *Changeability*: It should be possible to change test configurations and reschedule single measurements without restarting all clients

2. *Traceability*: The traffic resulting from each and every measurement should be recorded into a *pcap*-file. It should be possible to easily access the recorded traffic for a single test after the measurement is finished.
   This recording mechanism guarantees the traceability of the results. Additionally, if a property of the traffic should be measured that was not planned in the initial design, the property could also be reconstructed from the recorded traffic for all previously conducted measurements.

3. *Failure-Resistance*: In case of a failure, there should be some sort of fail-over mechanism that allows a recovery of the system.

4. Single tests should also be available in a format that allows effective evaluation while still containing all information necessary to evaluate them over a long-term time period.

In general, these tools were used:

Python 3 As a programming language used for almost all newly written code

MongoDB As a database management system for storing all measurement results. Since the result structure varies greatly for each test type, the *JavaScript Object Notaion* (JSON) is used as a data structure for the results. This allows more effective evaluations compared to *pcap* files while still retaining all information deemed relevant at time of the test design. MongoDB natively stores data in JSON and provides powerful queries for evaluating results.

Wireshark The *dumpcap*-tool provided by Wireshark is used to record network traffic. Wireshark's command-line tool *tshark* is used in the VoIP test to extract communication streams and their Quality of Service parameters from network traffic dumps.

dnspython This tool is used for conducting DNS tests directly in Python.

Scapy Used as a framework to craft custom network packets and send them over a network. Also, the PCAP parser provided by Scapy is used for evaluating recorded network traffic.

Tor OONI The Tor OONI-Framework is used for testing network interference and blocking of Tor traffic.

Bottle/Jinja2/Bootstrap/jQuery For providing a user-friendly maintenance frontend to the server.

Besides these tools, some other Python packages were used, providing some functionality necessary for smaller parts, e.g. *croniter* for parsing crontab-lines, *watchdog* for watching text files for changes or *requests* for easier HTTP communication between client and server.

The resulting code and software architecture was created using these tools and is based on the requirements mentioned above. Accordingly, tasks were split between server and client.

The server is used specifically to:

- Manage test configurations and schedule tests: The client polls every x minutes for new tests and the server. The server then sends a list of scheduled test types to the client, together with their respective configurations and time spans in which the test has to be executed.

- All test configurations are stored at the server in a JSON file. For the scheduling of the tests, a crontab-like schedule is given. Based on this schedule, the server assigns the tests to the single clients. The configuration used for testing is listed in Appendix A.

- Store all test results from the clients centrally in a database.

- Provide a management interface.

The client, in turn, has no own scheduling logic, but only conducts the tests given by the server. After the test is finished, the client sends the JSON results to the server.

The measurement framework on both client and server is kept simple. The following tasks are done by the framework on both client and server:

- *Schedule*: Individual tests on the client and their server counterparts are scheduled for startup and shutdown. The tests and test servers are represented as Python classes; the code is executed in threads. The shutdown methods are also called by the framework, regardless of the success of a test. This assures that in case of failure, resources such as ports are freed after task completion.

- *Record*: All network activity for each single test is recorded on both client and server into a *pcap* file, using *dumpcap*. This recording mechanism is directly integrated into the measurement framework and guarantees traceability of all results.

- *Store*: Test results in JSON format are sent from the client framework to the server, which in turn stores them into the database. Traffic dumps are not automatically transferred to the server but are manually backed up in regular intervals from the clients due to their large file size, often exceeding multiple hundred megabytes for traffic-intensive measurements.

3.2.3 Scheduling

A large part of the framework, as mentioned above, is the scheduling of tests. Scheduling is non-trivial since individual tests have different prerequisites:

- *Resources*: Some tests use the same resources, meaning that there can never be multiple tests that run at once while using the same port.
- *Time*: Some tests require more time than others. E.g. while the POP3 test *POP37* can be done in a matter of seconds, other tests require more time. This is especially important when connection speed is measured, which is done by *TCPS4* and *MM7*.
- *Used traffic*: This often correlates to the *time*-criteria: Some tests require more network traffic for measurements. There is an incentive to run these tests less often on a metered connection.

The first idea was to simply run tests in a queue. This was unsuccessful due to several reasons: For example, in case a client fails during a test, the server may enter a deadlock since it waits for a client result that is never transmitted. The same applies if a client loses its internet connection during a measurement. Since this happens once or twice a day due to IP changes and forced provider disconnects, a deadlock would be eventually inevitable. A queue also makes restarting of clients and servers more difficult, since at the time of a server restart, the queue information may be lost.

But the idea of conducting tests in a repeating fashion stuck. In the final scheduling framework, a crontab-like schedule together with a test duration was used. Using this, each test has a fixed assigned timeslot, in which it can be executed. This allows for clients to fail, since the server does not wait for responses but autonomously starts up and shuts down the individual measurement servers at the begin and at the end of the timeslot. It also allows for client restarts, since at every time it is known which tests are scheduled for future measurements. This also allows for different intervals, e.g. a short test can be executed multiple times per hour, while data-intensive tests are executed only a few times per day. Since timing is important for this matter, as the test slots are described by their UTC timestamps, all clients and the server's clock are synced via NTP.

As for resources, the tests are arranged in the schedule in a way, that there is a few seconds buffer between tests using the same port. So, even if there is some failure or timing inaccuracy, measurements are not affected. The resulting schedule can be seen in Table 3.1.

3.3 Metrics

In this section, all the used metrics are described. For each metric, a reason is given why it is important to measure in this specific way. Also, the configuration options are given. Naming of the metrics is done in the following scheme: *[protocol][osi-layer]* where *protocol* stands for the protocol that is tested, e.g. *TCP* or *HTTP* and *osi-layer* gives a numeric representation of the layer this test takes place in the OSI model[40].

Test	Start at min.	Duration (min)
cm7	0	3
vs7	3	1
http7	4	1
tcp4	5	2
udp4	7	2
tcps4	9	10
mm7	19	10
syn4	29	1
stls7	30	1
smtp7	31	1
pop37	32	1
voip7	33	3
tls4	36	1
ndns7	37	1
bdns7	38	1
trac3	39	1
ooni7	40	10

Table 3.1: Overview of test scheduling

3.3.1 Basic TCP measurements (TCP4)

This metric is designed to detect basic port blocking and different treatments of packets sent using the *Transmission Control Protocol*. TCP[29] is a protocol defined on the transport layer of the OSI model. It allows a reliable communication between endpoints and ensures that messages are received by the client application reliable, meaning that there are no out-of-order-messages, lost or duplicated segments. However, this entails that applications cannot access incomplete messages, rendering the protocol unsuitable for many real-time applications, e.g. online gaming or VoIP.

In this metric, a few TCP segments on a single port are sent from the client and server and returned by the server. The input configuration for this test is given by the JSON given in Listing 1. The configuration takes a list of ports that should be tested and a parameter of how often a test should be conducted for each port. The TCP connection is established only once for each port to minimize inaccuracies caused by the TCP handshake and connection establishment.

The tested ports correspond to the protocols[8] listed below:

- *20*: File Transfer Protocol (FTP)
- *80*: Hypertext Transfer Protocol (HTTP)
- *443*: Hypertext Transfer Protocol over TLS/SSL (HTTPS)
- *554*: Real-Time Streaming Protocol (RTSP)
- *1214*: Kazaa - Peer-to-peer file transfer

- *1725*: Valve Steam Client - online gaming
- *5060*: Session Initiation Protocol (SIP) - used for Voice over IP
- *6881*: BitTorrent - Peer-to-peer file transfer
- *8333*: BitCoin - cryptocurrency
- *48123*: Reference port, not used for any protocol

A TCP connection is established using a three-way handshake. The segments in the handshake make use of the *SYN*, *ACK*, and, in case of an unsuccessful attempt, the *RST flag*. Along with this, the *sequence number* and *acknowledge number* are used. In this handshake, the client initializes a connection by sending a segment with the *SYN*-flag set and arbitrary sequence number to the server. The server then responds by sending a segment with set *SYN* and *ACK*-flags, using its own arbitrary sequence number and increasing the client's sequence number by one and setting this as an acknowledge number. The client then finishes the handshake by sending a segment with the *ACK*-flag set to the server, acknowledging the server's sequence number.

In the implementation, it was originally planned to implement the connection including a full TCP handshake using *scapy* for finer-grained measurements. This plan was dropped in favor of using the standard Python network stack, since the performance of scapy was not satisfactory. Also complicating is that, by default, the Linux kernel handles all TCP connections. As a consequence, if another program uses a Raw socket to establish a TCP connection that the kernel is unaware of, the kernel sends a *RST*-segment as soon as it receives the *SYN-ACK* answer from the server, since it is unaware of the *SYN*-segment sent by the application, therefore treating the *SYN-ACK*-segment as an error and responding by resetting the connection[1]. As an easy remedy, it is possible to block outgoing *RST*-segments by iptables, effectively blocking the Linux kernel from interfering.

```
1 {
2     "ports" : [ 20, 80, 443, 554, 1214, 1725, 5060, 6881, 8333,
      ↪ 48123],
3     "pings" : 5
4 }
```

Listing 1: Configuration of TCP4

The sequence of communication is illustrated in Figure 3.3. As it can be seen, for each port given in the configuration, the client initiates a connection to the server, sends a *PING*-message, to which the server responds with a *PONG*. After receiving this message, the client responds to the server with *OK*. The server then sends its measured round-trip time to the client.

Measurements made by this metric are:

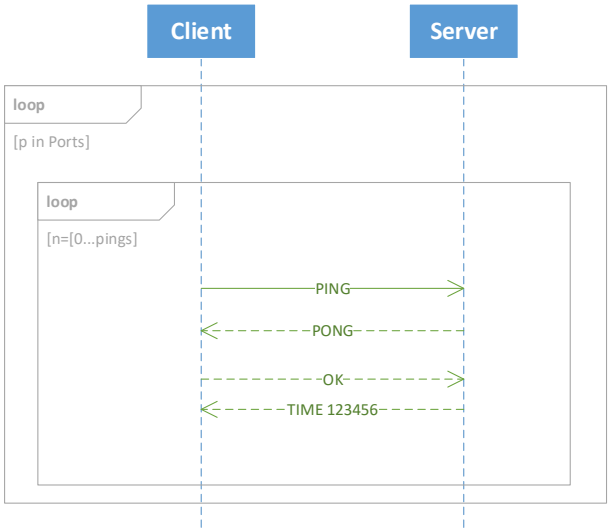

Figure 3.3: TCP4 sequence diagram

- If all segment are fully received by both client and server or if network operators block traffic.
- The round-trip time (RTT) from client-side and server-side. The client RTT is defined as the time that passes from sending the first *PING* segment until receiving the *PONG* segment from the server. The server RTT is defined as the time passing between sending the *PONG* segment and receiving the *OK* segment.
- The TTL of all packets received by the client. This is done to detect middleboxes that modify the TTL of packets in TCP communication on specific, but not all, ports.

The result of a measurement is described in Listing 2. It features a full list of all measured TTLs and RTT for every given segment. Also, the minimum, maximum, mean and median is given for the TTLs and RTTs.

3.3.2 Basic UDP measurements (UDP4)

Like the *TCP4* metric, the *UDP4* metric is designed to detect basic port blocking and middleboxes applied on specific ports by ISPs. The *User Datagram Protocol*[28] is also located at the transport layer in the OSI model. In contrast to TCP, it does not ensure reliable communication. When using UDP, it is therefore possible that packets arrive

```
{
    "220": {
        "ttl_source": [ 53, 53, 53, 53 ],
        "pings": [
            {
                "server": 352.116346,
                "begin": 1473156307,
                "client": 179.92615699768066
            },
            {
                "server": 187.517642,
                "begin": 1473156308,
                "client": 195.98698616027832
            }
        ],
        "ping_server_mean": 238.1682867,
        "ping_client_median": 220.94297409057617,
        "ttl_source_mean": 53,
        "ttl_source_median": 53.0,
        "ttl_source_max": 53,
        "ttl_source_min": 53,
        "ping_server_median": 206.28535649999998,
        "ping_client_mean": 261.25638484954834
    }
}
```


Listing 2: Result of a TCP4 measurement for port 220

out-of-order, in duplicates or not at all. This also ensures short transmission delays, rendering the protocol ideal for real-time applications like VoIP or online gaming.

The UDP4 metric follows the concept of the TCP4 metric. The configuration given in Listing 3 is identical – it only specifies the tested ports and how often a round-trip should be performed.

```
{
    "ports": [
        1725, 5060, 6881, 9987, 48123
      ],
      "packets": 10
}
```

Listing 3: Configuration of a UDP4 measurement

The test has to operate differently since UDP does not ensure that all datagrams are transmitted reliably. To tackle this, all messages sent between client and server contain a numeric identifier that allows assigning timings to the correct datagrams. This is shown in Figure 3.4.

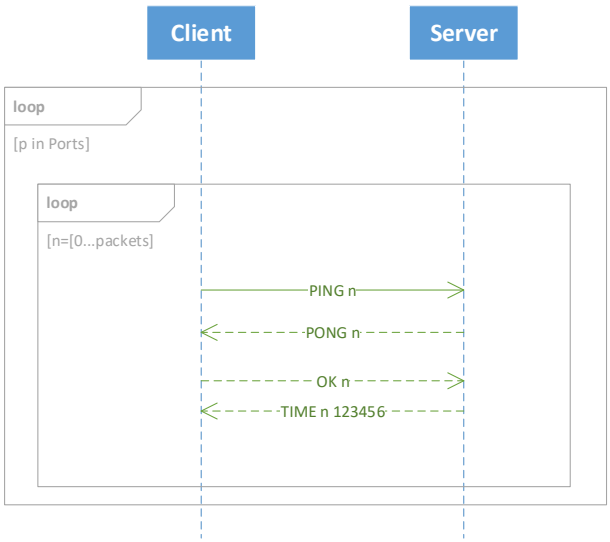

Figure 3.4: UDP4 sequence diagram

- If all datagrams are fully received, the RTT for client and server (see the section for the TCP4 metric) and the TTL values for all packets received by the client.
- Additionally: The rate of lost datagrams, and if datagrams are received out-of-order.

The result of a single measurement of the UDP4 metric is described in Listing 4.

3.3.3 SYN Flooding attack test (SYN4)

The SYN4 test simulates a *SYN Flooding attack*[11]. In a SYN Flooding attack, the attacker sends a great amount of TCP segments with a set *SYN flag* to the victim. The victim, in turn, returns a TCP segment with a set *SYN and ACK flags*, signaling that it is ready for a connection. The attacker, in turn, never answers sends a TCP segment with a *ACK flag* therefore never establishing the connection. Since the victim still waits for the attacker's answer, the resources remain blocked for a certain period of time. Using a large enough number, the attacker can block all of the victim's resources, causing a Denial-of-Service.

```
{
  "6881": {
    "ping_server_mean": 3.951,
    "ping_client_mean": 3.820,
    "ping_server_median": 4.16,
    "ping_client_median": 3.820,
    "pings": {
      "0": {
        "client": 3.289937973022461,
        "server": 4.166603,
        "begin": 1465227634.8416238,
        "received_as": 2
      },
      "n-1": {
        "client": 2.968311309814453,
        "server": 2.586364,
        "begin": 1465227634.8404565,
        "received_as": 8
      }
    }
  }
}
```


Listing 4: Result of a UDP4 measurement of port 6881

The idea behind this metric is to measure if ISPs allow SYN-flooding attacks originating from their customers.

The configuration of this test, listed in Listing 5 once again, is simple. It allows only to set the port range of the originating segments from which ports are randomly selected, the victim's port and the count of SYN segments that should be sent. It is implemented using the *scapy* library.


```
{
  "port": 443,
  "count": 200,
  "source_port": {
    "min": 40234,
    "max": 41453
  }
}
```


Listing 5: Configuration of a SYN4 measurement

The sequence of this test illustrated in Figure 3.5 is also straight-forward: The client sends n TCP segments to the server, the server responds with a *SYN-ACK* segment. The client then counts how many segments were received during the test. If the ISP filters the attack, either the SYN segments are never received by the server and the client therefore never receives SYN-ACK segments from the server. This is also represented in the result structure provided in Listing 6: It is saved how many SYN-ACKs were received by the client, along by the concrete origin ports used in the attack.

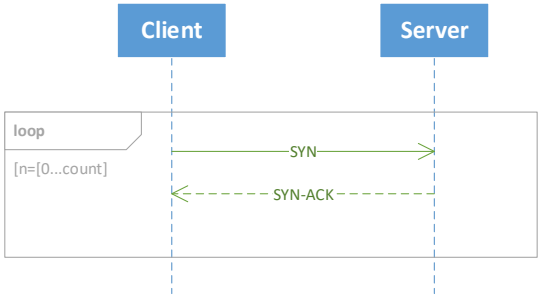

Figure 3.5: SYN4 sequence diagram

```
{
    "answers": 200,
    "used_ports": [
        40279,
        40893,
        41073
    ]
}
```

Listing 6: Result of a SYN4 measurement

3.3.4 Blocked Hosts DNS test (BDNS7)

The Domain Name System (DNS)[24] is a decentralized hierarchical Internet service, allowing clients to resolve hostnames (e.g. *tuwien.ac.at*) to the corresponding IP addresses. These DNS resolvers are typically provided to customers directly by the ISPs, meaning that every DNS request is handled by the customer's ISP in a first step.

In some countries, including Austria, certain domains are blocked for legal or political reasons. In Austria, the Supreme Court (*OGH*) made it clear in judgment *4 Ob 71/14s*, following judgment *C-314/12* of the European Court of Justice (*ECJ*), that ISPs can be

forced to block websites providing illegal content. In the case of the judgment, this was the website *kino.to*, but other websites followed shortly.

The court did not specify, in which way the websites have to be blocked. In practice, Austrian ISPs use their DNS nameservers to accomplish this. In the blocked DNS test (BDNS7), it is measured, if certain websites are blocked via DNS and in which way this censorship is implemented. For this, the test clients facilitate DNS requests using the *dnspython*-package and save the results. The configuration for this test can be found in Listing 7 and contains the hostnames that should be queried and the nameservers used. If there is no nameserver given, the default nameservers are used. The nameserver *8.8.8.8* is provided by Google[16] and serves as a control request. If the request with Google returns the domain, but the ISPs DNS does not, it can be assumed that a domain is blocked and not simply non-existent.

```
{
"requests": [
    { "host": "www.thepiratebay.se" },
    { "host": "www.kinox.to" },
    { "host": "www.123hjaf9hu32iufhuihoafine.com" },
    {
        "host": "www.kinox.to",
        "nameservers": ["8.8.8.8"]
    }
  ]
}
```

Listing 7: Configuration for both the BDNS7 and NDNS7 measurements

A result can be seen in Listing 8. Variables that are saved include:

- The IP address(es) provided by the nameserver.
- The return code provided by the DNS service, for being able to distinguish between non-existing hosts, existing hosts and timeouts.
- The runtime of the request.

3.3.5 Non-existing Hosts DNS test (NDNS7)

The DNS test for non-existing domains follows the same configuration (Listing 7), implementation and result structure (Listing 8) as the BDNS7 test. The objective, however, is to measure, how providers handle non-existing domains. The measurement was implemented following the paper of *Weaver et al.* [35], who found that some providers use(d) custom error pages, featuring auto-correction of domains, advertisements and other content.

```
{
  "duration_ms": 84.296227,
  "host": "www.123hjaf9hu32iufhuihoafine.com",
  "rcode": 3
},
{
  "rcode": 0,
  "duration_ms": 14.236689,
  "ttl": 3600,
  "entries": [ "0.0.0.0" ],
  "host": "www.kinox.to"
},
{
  "host": "www.kinox.to",
  "rcode": 0,
  "nameservers": [ "8.8.8.8" ],
  "duration_ms": 38.822174,
  "ttl": 299,
  "entries": [ "104.28.21.67", "104.28.20.67"
  ]
}
```


Listing 8: Result for both the BDNS7 and NDNS7 measurements, containing blocked and non-existent domains

3.3.6 HTTP Caching and Manipulation (CM7)

In this test, caching and manipulation of content delivered over the Hypertext Transfer Protocol (HTTP)[12] is measured. This test is based on the papers of *Nakibly et al.* [26] and *Xu et al.* [37], who found that some network operators manipulate ad content for monetization and cache objects for performance increases. The CM7 test works by simulating HTTP traffic, as illustrated in Figure 3.6. It measures for two possible alterations done by ISPs:

Caching It is measured if ISPs cache some website resources. For this, the same unencrypted HTTP request is sent multiple times to a server. Since the server always responds with the HTTP Header field *Cache-Control: max-age=600, public*, the client is (in theory) allowed to cache the resource and not query the server again on future requests[12]. So, in theory, ISPs could also cache this resource and directly deliver it to customers, reducing network traffic. If this is the case, HTTP requests sent by the client should never reach the server.

Manipulation It is measured if an ISP manipulates content. For this, checksums of every HTTP response header/body are calculated and compared. If there is no

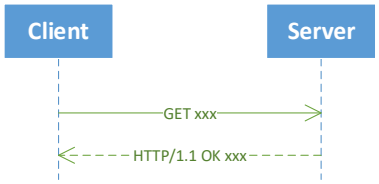

Figure 3.6: CM7 sequence diagram

manipulation, these checksums have to be identical on both client and server. The motivation for ISPs to manipulate content can be manifold: For example, ISPs could replace ad identifiers with their own, generating additional revenue. On the other hand, ISPs could optimize files (e.g. by applying lossless compression) and optimize network performance.

The configuration in Listing 9 is therefore held flexible: All requests that are sent during a test are specified, including which HTTP headers should be used and if the requests should be made multiple times. Some additional header fields are set directly in the Python code.

To measure caching, the number of sent requests is compared with the number of requests received by the server, as demonstrated in the result data structure in Listing 10. For measuring manipulation, the checksums of both header and body of the HTTP response are generated using the *SHA-256* hashing algorithm and compared for equality.

3.3.7 HTTP Antivirus test (VS7)

The goal of the VS7 metric is to detect if ISPs use deep package inspection to apply virus-protection for end customers. For this, an EICAR test file is requested via an unencrypted HTTP request and transmitted to the client.

The *European Institute for Computer Antivirus Research* (EICAR)[36] provides this test file as a tool to test antivirus software. As such, the file is recognized as a virus by almost all software products[34]. It contains no functionality but produces correct output when run as an executable.

The configuration, result and sequence of the test are identical to that of the CM7 test: The client requests the EICAR test file from the server using HTTP, the server responds with a valid answer for this file. Once again, it is measured, if the response from the server is received without modifications by the client. For this, again, the SHA-256 hashing algorithm is used to assert equality of the received file.

```
{
  "requests": [
    {
      "resource": "GET
       ↪ /959ffdfd-1c56-4294-97e4-397c0e1e4171/image2.jpg
       ↪ HTTP/1.1",
      "repeat": 3
    },
    {
      "header": [
        "Host: r1---sn-4g5edne7.googlevideo.com",
        "User-Agent: Mozilla/5.0 (Windows NT 10.0)
         ↪ Gecko/20100101 Firefox/47.0",
      ],
      "resource": "GET
       ↪ /videoplayback?mime=video/webm&upn=q_PY3To1fWI
       ↪ HTTP/1.1"
    }
  ]
}
```

Listing 9: Configuration for a CM7 measurement

3.3.8 Invalid HTTP syntax test (HTTP7)

Besides VS7 and CM7, the HTTP7 test also is aimed at detecting middleboxes applied to HTTP traffic by ISPs. For this, invalid HTTP requests and responses are sent by both client and server. This technique is also used by the Tor OONI project[38]. In the Tor OONI-project, this lead to the detection of the use of *BlueCoat*, *Squid* and *Privoxy*-middleboxes in eleven countries.

The configuration, result and sequence of the HTTP7 test are once again identical to their counterparts in the CM7 test. However, in generating results, the focus is more on the integrity of the HTTP header fields rather than at the HTTP bodies. If there are proxies used by the ISPs, these faulty requests would in some cases either never reach the server, or be auto-corrected by the used proxy.

3.3.9 Voice over IP test (VOIP7)

In the Voice over IP test, the quality of VoIP-streams is measured. Especially for mobile ISPs, VoIP can be harmful to their business model[6], as customers may use the Internet to conduct telephony rather than billed connections. This is especially true for overseas connections. If connections are not blocked altogether, it can seem lucrative to degrade the quality of the VoIP connection, leading customers to use regular billed connections.

```
{
    "matches": [
        {
            "request": "GET
             ↪ /959ffdfd-1c56-4294-97e4-397c0e1e4171/image2.jpg
             ↪ HTTP/1.1",
            "header": {
                "server": "6834c6a767da721[...]8d0058976",
                "client": "6834c6a767da721[...]8d0058976"
            },
            "body": {
                "server": "275a021bbf[...]4538aabf651fd0f",
                "client": "275a021bbf[...]4538aabf651fd0f"
            }
        }
    ],
    "leftover": [],
    "mismatches": []
}
```

Listing 10: Result for a CM7 measurement

As the blocking of VoIP traffic is detected by other tests (namely TCP4 and UDP4), the VOIP7 test measures different Quality of Service (QoS) metrics for VoIP connections. For this, a call is simulated using a replay of a pre-recorded Real-Time Transport Protocol (RTP)[30] stream. The prerecorded call is given in a pcap-file. The individual packets are loaded, parsed and filtered using scapy's *scapy.all.readpcap* functionality. The payload itself is then sent using the standard Python UDP socket implementation.

The sequence of this measurement is shown in Figure 3.7. While replaying the stream, all traffic is recorded on the client-side. After the call is finished, *tshark*, the command-line variant of Wireshark, is invoked to give QoS metrics for the connection using:

```
tshark -r pcap_file.pcap -d udp.port==2222,rtp -q -z rtp,streams
```

The input configuration for the VOIP7 measurement in Listing 11 consists only of the pcap-file containing the call that should be replayed and the duration of the call. The output of tshark is then parsed and used for the JSON-result given in Listing 12. Good measurements for the QoS are especially the jitter, packet loss and delta metrics provided by tshark.

3.3.10 Malformed TLS handshake test (TLS4)

Transport Layer Security (TLS)[9] provides encryption, mainly for TCP connections. The secure channel provided by this software stack is established using a handshake

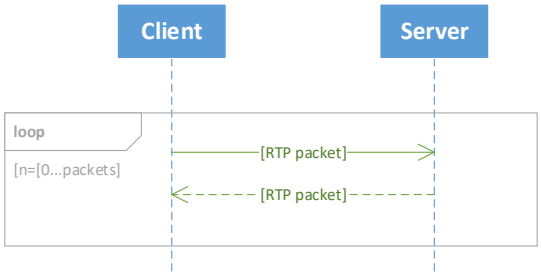

Figure 3.7: VOIP7 sequence diagram

```
{
  "ports": [
    2222
  ],
  "replay_pcap": "./metrics/voip7/rtpstream.pcap",
  "test_uuid": "77b26c92-27a7-49dd-846c-fc20906a0c80",
  "call_duration_ms": 14000
}
```

Listing 11: Configuration for a VOIP7 measurement

Byte offset	0	1	2	3	4	5	6	7	8	9..n
Description	record type	TLS version major	minor	length	length	client hello	length	length	length	message
Example	0x16	0x03	0x01	0x01	0x0c	0x01	0x00	0x00	0xf7	0x03...

Table 3.2: TLS record containing a handshake

between client and server based on asymmetric cryptography. The intent of the TLS4 measurement is to find, if this handshake is intercepted and inspected by middleboxes. For this, a handshake is simulated, using syntactical errors, e.g. by setting by length fields that do not correspond with the actual length. While this is ignored by the custom code on both test client and server, middleboxes may throw errors or do not allow to establish connections at all. The sequence of this test is illustrated in Figure 3.8.

An example for these length fields is listed in Table 3.2. The byte sequence contains the first 10 bytes of a TLS record representing a handshake (record type 0x16). The record contains a "client hello" as part of the handshake (type 0x1). Bytes 2 and 3 state the length of the record including the handshake, bytes 6, 7 and 8 state the length of the handshake record contained in the TLS record. In the test, these bytes representing the length of the record are replaced by random data, invalidating the TLS record in the process.

```
{
    "statistics": [
        {
            "jitter_ms_max": 8.43,
            "packets": 9,
            "dst_ip": "10.0.0.8",
            "dst_port": 57478,
            "jitter_ms_mean": 19.45,
            "payload": "PCMU",
            "loss_percent": 0.0,
            "src_ip": "128.130.204.34",
            "loss_packets": 0,
            "ssrc": "0x40C735AF ITU-T G.711",
            "delta_ms_max": 109.31,
            "src_port": 2222
        }
    ]
}
```


Listing 12: Result for a VOIP7 measurement

The test needs no further configuration since a valid handshake is pre-loaded and made invalid with random mutations in different length fields. The result shown in Listing 13 therefore only contains the matching and mismatching SHA256-checksums of the handshakes sent and received.


```
{
    "matches": {
        "server": "26d361eb15fd05c5937[...]c5e7b0d82c8",
        "client": "26d361eb15fd05c5937[...]c5e7b0d82c8"
    }
}
```


Listing 13: Result for a TLS4 measurement

3.3.11 Invalid POP3 syntax test (POP37)

Similar to HTTP7 and TLS4, the POP3 test tries to detect the usage of middleboxes by transmitting traffic containing syntactical errors. The Post Office Protocol in version 3 (POP3)[25] allows users to retrieve emails from mail servers using a TCP connection. During this process, the account credentials are transmitted. Based on this, the test uses a malformed email address as input data, making the authentication process invalid. The

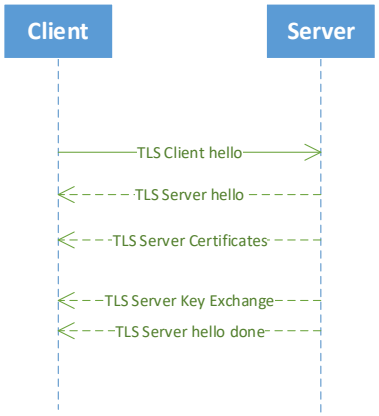

Figure 3.8: TLS4 sequence diagram

schematic for this test can be seen in Figure 3.9. The test is repeated for multiple ports, in case of middleboxes only inspecting traffic on specific TCP ports.

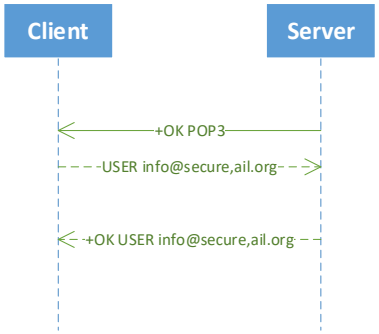

Figure 3.9: POP37 sequence diagram

The test takes no special configuration other than the ports that should be used, the result shown in Listing 14 simply states if the malformed address was received by both client and server.

```
{
  "110": {
    "invalid_address_received": true
  },
  "8110": {
    "invalid_address_received": true
  }
}
```

Listing 14: Result for a POP37 measurement

3.3.12 Invalid SMTP syntax text (SMTP7)

The SMTP7 measurement is similar to the HTTP7, POP37 and TLS4 tests as it again tries to detect hidden middleboxes by issuing malformed traffic, this time for the Simple Mail Transfer Protocol (SMTP). SMTP[22] allows users to send electronic mail. The protocol uses TCP port 25 as a standard port, allowing users to send mail with or without authentication.

Once again, malformed requests are sent, as can be seen in the sequence diagram shown in Figure 3.10. The configuration given in Listing 15 allows to turn on/off these syntax errors. The result structure in Listing 16 simply states if the content integrity is given, meaning that the content that was sent by the client reached the server without any modifications.

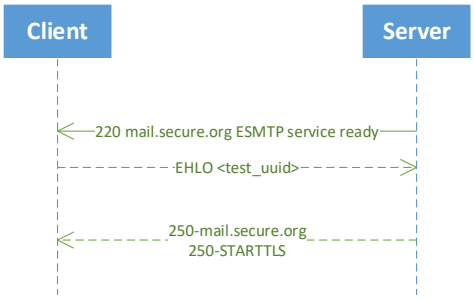

Figure 3.10: SMTP7/STLS7 sequence diagram

3.3.13 StartTLS stripping test (STLS7)

StartTLS[17] provides secure SMTP communication by securing connections with Transport Layer Security. For this, during the SMTP communication, the server lists a flag,

```
{
  "ports": [ 25 ],
  "valid_response": false/true
}
```

Listing 15: Configuration for the SMTP7 and STLS7 measurements

```
{
  "25": {
    "content_integrity": true,
    "starttls_client_received": true,
    "starttls_server_received": true
  }
}
```

Listing 16: Result for the SMTP7 and STLS7 measurements

signaling the client that StartTLS is available for communication, offering the client a chance to upgrade to a secure connection.

In the past, some ISPs have been known for not transmitting this flag to the client[18], thereby effectively prohibiting their clients from establishing a secure connection for email sending, even though their email servers offer StartTLS. The STLS7 measurement aims to detect stripping of this flag. For this, an SMTP session is established between client and server. In case the ISP prohibits StartTLS, the server sends the StartTLS flag to the client which it never receives.

The configuration structure (Listing 15), the communication flow (Figure 3.10) and the result (Listing 16) are identical to the SMTP7 test. Only the configuration differs: The SMTP request from the client is answered with a syntactically valid response from the server.

3.3.14 TCP bandwidth test (TCPS4)

In the speed test over TCP, the connection speed over various TCP ports is measured. It aims to detect traffic shaping of ISPs for specific ports, e.g. port 6881 for BitTorrent traffic.

For performing a reliable speed test measurement while still controlling both test server and the test clients, an implementation of the *RTR Multithreaded Broadband Test* (RMBT)[33] is used. This specification allows for accurate measurements of download and upload speeds using a one or more concurrent threads. The Austrian website *www.netztest.at* is based on this specification and counts more than two million speed tests at the time of this thesis.

Since there was not yet an implementation of this protocol available in Python, an implementation of a stripped-down RMBT test was custom-made for this thesis. The

communication sequence illustrated in Figure 3.11 is identical to the communication of other RMBT-based tests. All data for the transfer during measurements is generated at random, using *os.urandom* which provides random data on-the-fly fast enough for all connection speeds. An RMBT test is divided into multiple phases:

1. *Initialization:* The test is initialized, the client is authenticated with the server.

2. *Download pre-test:* For getting a rough estimate of the client connection speed, a short download pre-test is conducted. This pre-test works by sending doubling amounts of data chunks for a given time. E.g. in the first iteration, 4096 bytes are send, in the second iteration 8192 bytes, then 16384 and so on, until the time limit of two seconds is reached. In other RMBT test clients, this is used to determine how many threads are used in the final measurement. Since the implementation of TCPS4 uses only one thread and rough client connection speeds are known, this phase was not implemented.

3. *Ping test:* The connection round-trip time is measured. For this thesis, this test is not featured in TCPS4. However, it was implemented in TCP4, which follows the RMBT protocol for conducting these ping tests.

4. *Download test:* The client requests the start of the download speed test by sending *DOWN n* to the server, where *n* represents a number of milliseconds that the test should last. The server then sends data chunks of a fixed size (4096 bytes in this implementation) to the client. During this period, the last byte of each chunk is set to *0x00*, signaling the client that the download test is not yet finished. After this period, the server sets the last byte of the last chunk to *0xFF*, signaling the client that the test is finished.

5. *Upload pre-test:* This phase mirrors the download pre-test: The client sends rounds of doubling data sizes to the server for a given period of time, determining a rough speed estimate. This phase was also not implemented for TCPS4.

6. *Download test:* This phase mirrors the download test phase: The client sends a continuous stream of data chunks to the server for a given time, masking their last bytes with *0x00*, except for the last chunk, which is masked with *0xFF*.

The test configuration is shown in Listing 17. It contains all ports that should be tested together with how long the single tests should last. If *concurrent* is set, all ports are tested at the same time. *Packet_size* gives the chunk size in bytes used for all tests.

The result in Listing 18 follows a stripped-down data structure an RMBT measurement. In the result, for each of the phases, download and upload, the measured duration and the transmitted bytes are given, allowing to calculate the connection speed. Additionally, roughly every 50 milliseconds, an interim value ("speed curve") is given, allowing to measure the connection speed over the time of a measurement.

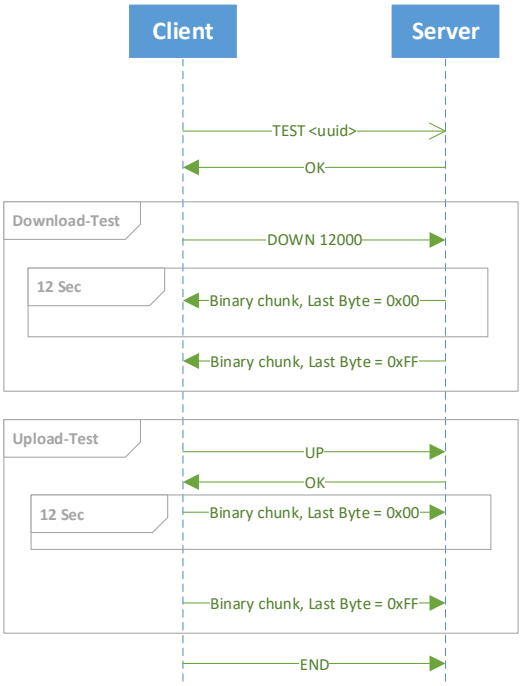

Figure 3.11: TLS4 sequence diagram, following the RMBT specification

Since the data structure is identical to the structure used by the RMBT-implementation of netztest.at, code reuse is possible. As an example, Figure 3.12 shows the speed graph during a measurement of TCPS4, generated with code provided by netztest.at.

3.3.15 Multimedia test (MM7)

The multimedia test is designed to detect traffic shaping that uses similar techniques to BingeOn used by T-Mobile USA[19]. T-Mobile USA shapes connection speed based on fields found in the *Host*-field found in the HTTP header.

For testing the speed, the RMBT implementation used in the TCPS7 measurement is used. It is only modified in two ways:

- There is no upload test since HTTP is stateless and there is no further upload in the connection, it the keep-alive header is not used.
- The test configuration transmitted from the client to the server, e.g. for signaling when the download test should start and how long it should take, is all reduced into a single request, mimicking a normal HTTP GET-request.

```
{
  "ports": [
        { "port": 80 },
        { "port": 443 },
        { "port": 6881 },
        { "port": 48123 }
      ],
      "packet_size": 4096,
      "test_duration_ms": 12000,
      "concurrent": false,
}
```

Listing 17: Configuration of a TCPS4 measurement

```
{
  "48123": {
    "upload": {
      "duration_ms": 12319,
      "bytes": 12312576,
      "speed_curve": [
        {
          "time_elapsed": 64,
          "bytes_total": 4096
        },
        {
          "time_elapsed": 120,
          "bytes_total": 8192
        }
      ]
    }
  }
}
```

Listing 18: Result of a TCPS4 measurement

The resulting communication sequence can be seen in Figure 3.13: The client initializes the test by sending an HTTP request, containing all relevant information, while retaining the *Host*-header field mimicking traffic from a video provider. The server responds with an HTTP status 200 (OK) and begins to execute the download test phase as specified in the RMBT protocol.

The configuration for an MM7 test is shown in Listing 19. It is similar to a TCPS4 test but allows to additionally specify the concrete HTTP headers that should be used in both the client request and the server response. The result is identical to the result of a

0.18 s	0.37 Mbps	8.2 KB
0.36 s	1.2 Mbps	37 KB
0.58 s	1.2 Mbps	70 KB
0.79 s	0.63 Mbps	86 KB
0.99 s	1.1 Mbps	110 KB
1.20 s	0.95 Mbps	140 KB
1.41 s	1.7 Mbps	180 KB

Figure 3.12: Visualization of connection speed during a TCPS4 measurement, generated by netztest.at-code

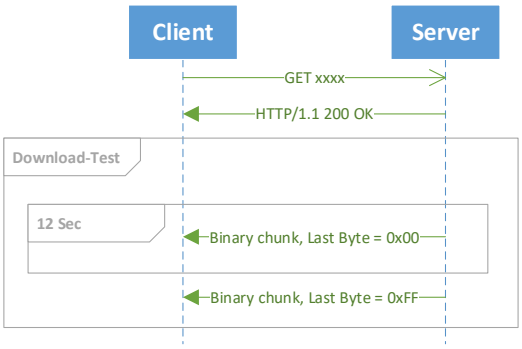

Figure 3.13: MM7 sequence diagram

TCPS4 test (see Listing 18) but misses the upload component, since no upload speed test is executed. It is still compatible with the data structure used by netztest.at, allowing the use of existing code for evaluations, e.g. the graphical representation of the interim speed given in Figure 3.12.

```
{
  "ports": [
    {
      "port": 80,
      "request_with": 1,
      "answer_with": 0
    }
  ],
  "http_headers": [
    {
      "id": 0,
      "header": "HTTP/1.1 200 OK\r\nAccept-Ranges: bytes\r\n"
    },
    {
      "id": 1,
      "header": "GET /videoplayback?mime=video/webm\r\n\r\n"
    }
  ],
  "test_duration_ms": 12000,
  "concurrent": false,
  "packet_size": 4096
}
```

Listing 19: Configuration of a MM7 measurement

3.3.16 Traceroute test (TRAC3)

Traceroute is a tool used to detect which intermediate nodes are used when establishing an Internet connection. As it is mentioned in the paper by *Xu et al.* [37], some providers use caches for certain websites for some times of the day. If this is the case in Austria, the intermediate connection changes for the same websites at different times of the day. Also, if the ISP optimizes the route for certain websites, this can be detected via a traceroute.

Traceroute works by sending IP packets with low *time to live* (TTL) values, beginning with one and adding one for each iteration. It the route to the target is longer than the TTL value, the intermediate host where the TTL reached zero, responds with an *ICMP* packet, informing the client that the TTL was exceeded during transit. Using these ICMP packets, the route to the target host can be reconstructed.

For performing traceroute, different protocols can be used, e.g. ICMP packets, UDP datagrams or TCP segments. In this implementation, TCP segments are used by scapy's implementation available in *scapy.inet.traceroute*. The test configuration shown in Listing 20 contains which hosts should be queried. A result, as listed in Listing 21, contains the route found for the individual hostnames, including all intermediate IPs and the TTLs that were used to find them.

```
{
    "requests": [
        { "host": "www.orf.at" },
        { "host": "www.google.at" }
    ]
}
```

Listing 20: Configuration of a TRAC3 measurement

```
{
    "requests": [
        {
            "host": "www.orf.at",
            "result": [
                { "ip": "192.168.8.1",      "hop": 0 },
                { "ip": "10.122.9.236",     "hop": 1 },
                { "ip": "10.122.8.9",       "hop": 2 },
                { "ip": "10.122.6.178",     "hop": 3 },
                { "ip": "194.232.104.149",  "hop": 4 }
            ]
        }
    ]
}
```

Listing 21: Result of a TRAC3 measurement

3.3.17 Tor OONI test (OONI7)

The Tor OONI framework[13] is a censorship analysis tool and allows for tests detecting ISP interference and blocking, all while using a Tor-secured connection as a reference connection. Since some websites in Austria are blocked by ISPs, it makes sense to utilize a pre-existing framework to detect censorship and obtain in-detail results of this censorship.

Tor OONI uses YAML for both its own configuration with *testdecks* and its results. Out of the many features of OONI, the only select modules were used. The complete testdeck is given in Appendix B. An overview is listed below:

- *manipulation/http_invalid_request_line* tests if malformed HTTP requests are manipulated and serves as a complimentary test to HTTP7.
- *manipulation/http_header_field_manipulation* tests if HTTP header fields are manipulated and serves as a complimentary test to HTTP7, CM7 and VS7.
- *blocking/web_connectivity* tests the blocking of *kinox.to* and *movie4k.to* and serves as a complimentary test to BDNS7. Additionally, it also records the actual response

of the blocked site, providing more additional information than just the DNS response found in the result of BDNS7.

During its tests, a Tor connection is established, therefore additionally testing for ISP interference with Tor.

OONI is invocated directly from the Python code, allowing control of input and output via pipes. The results are given in OONIs own YAML structure, which is encoded into a String in the resulting JSON, as shown in Listing 22.

The OONI framework is invocated using the following code:

```
ooniprobe -i testdeck.deck -n
```

Using the *-n* command line parameter ensures that results are not automatically transmitted to the Tor project, therefore keeping the measurements private.

```
{
    "results": {
        "report-http_invalid_request_line": "####...",
        "report-web_connectivity": "###...",
        "report-http_header_field_manipulation": "###..."
    }
}
```

Listing 22: Result of a OONI7 measurement

CHAPTER 4

Results

In this chapter, the results of all tests are listed. For each of the different measurements, a summary of the results is given.

4.1 Overview

Over the course of two months, over 100,000 measurement samples were gathered. The distribution of these tests over the providers and the different metrics can be seen in Table 4.1. There were some downtimes of clients as can be seen in Figure 4.2. These were partly caused by software problems, network failures, and hardware problems. Over the cause of the measurement period, two NUCs had to be replaced. Some of the network connectivity problems at first seemed to occur at random and were later attributed to loose contacts of the network cables, depicted in Figure 4.1.

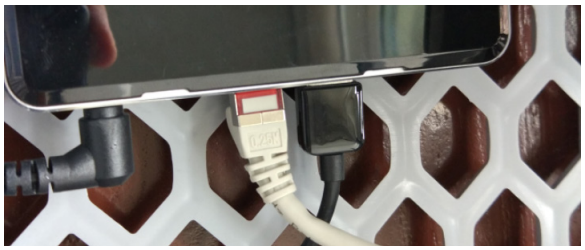

Figure 4.1: Loose contacts of a network cable (white)

4.2 Basic TCP measurements (TCP4)

The motivation behind the TCP4 measurement was to find shaping and blocking of traffic by ISPs based on TCP segment characteristics and differences in round-trip times (RTT).

Provider	bdns7	cm7	http7	mm7	ndns7	ooni7	pop37	smtp7	stls7	syn4	tcp4	tcps4	tls4	udp4	voip7	vs7	trac3	Total
A1 DSL	1897	1584	1706	1603	1902	1315	1845	1737	1800	1901	1557	1785	1885	1675	1733	1698	1899	29522
A1 LTE	1967	1678	1745	1579	1970	1418	1922	1757	1895	1918	1576	1810	1899	1779	1753	1751	1906	30323
Drei	1411	1403	1396	1374	1411	964	1412	1413	1412	1413	1072	1408	1170	1413	1412	1398	1410	22892
TMA	1805	1636	1721	1528	1807	1423	1799	1808	1808	1807	1473	1738	1807	1703	1807	1623	1807	29100
UPC	1061	1034	1025	990	1063	616	1065	1065	1064	1064	829	662	860	1065	1062	1036	1063	16624
Total	**8141**	**7335**	**7593**	**7074**	**8153**	**5736**	**8043**	**7780**	**7979**	**8103**	**6507**	**7403**	**7621**	**7635**	**7767**	**7506**	**8085**	**128461**

Table 4.1: Conducted tests during the measurement interval

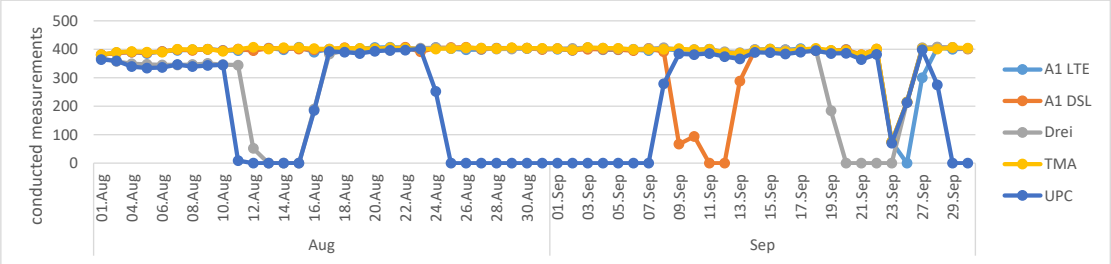

Figure 4.2: Measurement results over the months 08/2016-09/2016

4.2.1 TTL

In the test, the time to live property of each packet was recorded, as can be seen in Table 4.2 and Figure 4.3. While the TTLs of the tested ports were equal in general, there were two exceptions:

- UPC had a different TTL on port 554 (RTSP).
- T-Mobile had a different TTL on port 80 (HTTP).

While the difference with T-Mobile was significant (a median increase of 8), the difference with UPC was also consistent, but much smaller. At least for T-Mobile, the use of a deep packet inspection seems likely (as can also be seen in measurement CM7). For UPC, no manipulation of traffic could be detected - the TTL could, therefore, either origin from a middlebox of the ISP, but also from some treatment from the provided modem.

There were also some smaller inconsistencies with TTLs of other ISPs and other ports, but these were only negligible (e.g. only for a single packet during a measurement). Another interesting occurrence was, that while the fixed-line ISPs A1 DSL and UPC stayed consistent with their TTLs, the TTL values for mobile ISPs varied to a much greater degree.

4.2.2 RTT

Besides the TTL value, the round-trip time of packets was also measured. This was done to measure traffic shaping during some hours of the day. However, no significant

	Port 80	Port 220	Port 443	Port 554	Port 1725	Port 5060	Port 6881	Port 8333	Port 48123
A1 DSL	54,0000	54,0000	53,9970	54,0000	54,0000	54,0000	54,0000	54,0000	54,0000
A1 LTE	48,4554	48,5374	48,5322	48,5374	48,5374	48,4326	48,3598	48,3598	48,3598
Drei	51,3293	51,3293	51,3293	51,3293	51,3293	51,3293	51,3293	51,3293	51,3293
TMA	60,9939	52,9915	52,9936	52,9929	52,9929	52,9929	52,9932	52,9932	52,9932
UPC	54,9988	54,9988	54,9988	53,9988	54,9988	54,9988	54,9988	54,9988	54,9988
Overall	54,0289	52,1990	52,1977	52,0605	52,1993	52,1227	52,0933	52,0933	52,0933

Table 4.2: Mean TTL of the tested ports in TCP4

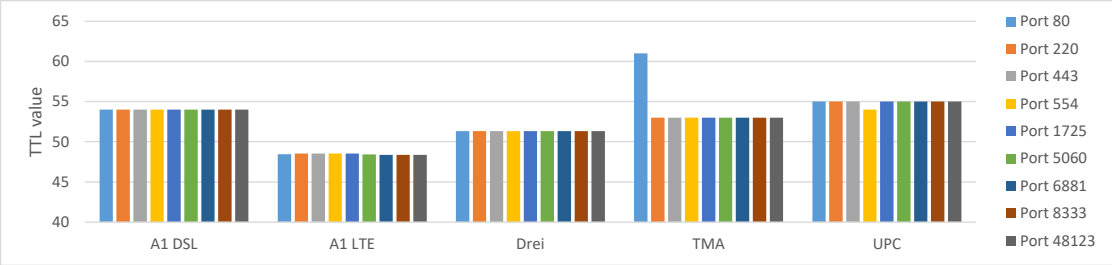

Figure 4.3: Average TTL values as given in Table 4.2

differences in RTTs were measured, neither depending on the protocol nor depending - as the test was conducted hourly - on the time of the day.

4.3 Basic UDP measurements (UDP4)

Similar to TCP4, for the UDP4 test, the round-trip times were measured and the TTL values recorded.

When inspecting the TTL values, it is obvious that TTL values vary more greatly than with TCP measurements as can be seen in Table 4.3 and Figure 4.4. This, however, does not entail that traffic is treated differently, as not all ports were tested successfully in all measurements.

Also, port 5060 (SIP - Session Initiation Protocol) seemed to be blocked for A1 LTE customers. If this holds true for all customers or only for the test client depending on a specific hardware/software environment could not be determined.

Again, there could not be measured any difference in RTTs depending on the time of the day or the used port.

4.4 SYN Flooding attack test (SYN4)

Out of over 6,500 individual measurements sending 20 SYN packets, only in less than 1% packets were filtered. It is therefore concluded that no ISP filters SYN-flooding attacks, at least not when only sending 20 packets.

	Port 1725	Port 5060	Port 6881	Port 9887	Port 48123
A1 DSL	60,3432	59,4993	60,2568	59,4854	57,6820
A1 LTE	59,7595		58,8332	57,9796	55,4921
Drei	61,5821	60,1178	61,0462	60,0675	57,0763
TMA	60,3002	58,3947	59,5200	58,8313	57,0600
UPC	61,2847	60,9183	61,1495	60,5114	58,3477
Overall	60,5751	59,5867	60,0245	59,1972	56,9565

Table 4.3: Mean TTL of the tested ports in UDP4

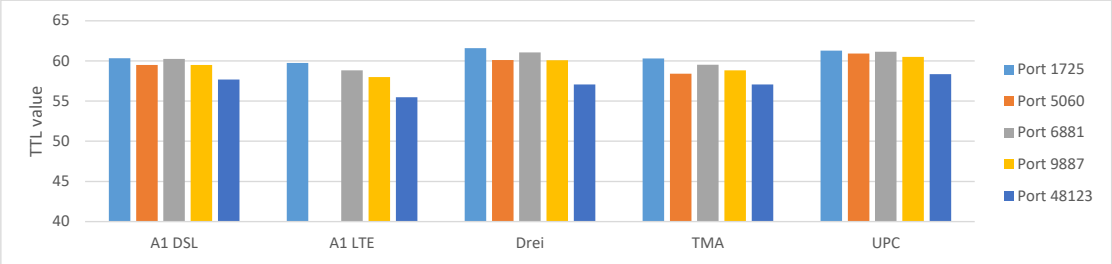

Figure 4.4: Average TTL values as given in Table 4.3

4.5 Blocked Hosts DNS test (BDNS7)

The tested domain *www.kinox.to* was blocked by all ISPs. The domain *www.thepiratebay.se* was not blocked by T-Mobile, but blocked by all other providers. The censorship was done using DNS. Interestingly, not a single provider just returned a *NXDOMAIN* record, signaling that the domain does not exist. Instead, ISPs redirect traffic to pages informing users about the censorship. An exception to this is UPC, which redirects users to the non-existent IP *0.0.0.0*.

The resolved IP addresses can be seen in Table 4.4. The IP addresses were consistent throughout the measurement interval. Interestingly, *thepiratebay.se* is not blocked by T-Mobile Austria, as the request resolves to IP addresses belonging to the *CloudFlare* Content Delivery Network. The pages informing about the censorship of *thepiratebay.se* and *kinox.to* are identical for each provider with the exception of A1 LTE, which uses two different pages, giving different reasons. For *kinox.to* it is stated that the website is blocked due to a court order (*"einstweilige Verfügung des Handelsgerichts Wien"*) while the website for *piratebay.se* states the reason as a censorship request based on the Austrian copyright law (*"Sperrbegehren nach § 81 Abs 1 UrhG"*).

The pages informing about the censorship are publicly accessible without a *Host*-field in the HTTP header for all providers with the exception of Drei. For Drei, the IP address is only reachable from within the Drei network. Also, a response is only generated, if a valid *Host*-header field is transmitted. The pages can be reached using *netcat*:

```
$ nc 213.94.80.154 80
```

Host	www.thepiratebay.se	www.kinox.to
A1 LTE	213.33.66.164	213.33.66.163
A1 DSL	213.33.66.164	213.33.66.164
Drei	213.94.80.154	213.94.80.154
TMA	104.31.10.172, 104.31.11.172	212.166.122.119
UPC	0.0.0.0	0.0.0.0

Table 4.4: IP resolution for blocked pages by Austrian ISPs

	www.123hjaf9hu32iufhuihoafine.com	Duration (ms)
A1 LTE	Timeout	5005,17
A1 DSL	Not found	81,89
Drei	213.94.80.190	158,92
TMA	Not found	1261,49
UPC	Not found	133,97

Table 4.5: IP resolution for non-existing pages along with the mean duration of the resolution request

```
GET / HTTP/1.1
Host: www.kinox.to

HTTP/1.0 302 Found
Location: http://www.drei.at/portal/de/privat/info/sperre.html
Server: BigIP
Connection: Keep-Alive
Content-Length: 0
```

4.6 Non-existing Hosts DNS test (NDNS7)

In this measurement, a DNS request for the non-existing domain *www.123hjaf9hu32iufhuihoafine.com* was facilitated using the default DNS nameservers provided by the ISPs. The request was resolved as shown in Table 4.5. The resolved IPs and return codes were consistent throughout the measurement period.

T-Mobile, UPC, and A1 on landline connections resolved correctly, issuing a *NXDOMAIN* return code. A1 on LTE connections did not return anything but timeout after the maximum time of 5 seconds allowed by the measurement.

The exception to this is Hutchison Drei, which returned a page featuring Google Analytics and a Google Custom Search form. This search form is automatically transmitted when loading the page, searching for the non-existing host name. The IP, as in the BNDS7 measurement, once again could only be reached from within the Drei network and needs a transmitted *Host* header field. The HTML of this page can be seen in Listing 23.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  ↪  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>Seite nicht gefunden</title>
        <style>
                body    {margin: 0; padding: 0;font-family: Arial, Helvetica, sans-serif;
                  ↪  overflow-x: hidden;min-width:980px;}
                header, section, article        {display: block;}
                .header {background: black
                  ↪  url(http://live-pse-download.drei.com:8000/UN/content/dns/bgHeader.png) 0
                  ↪  100% repeat-x;padding: 20px;height: 40px;}
                h1      {margin: 10px 0;color:white;font-size:large;}
                .logo3er        { float: left; margin-right: 20px;}
        </style>
        <link rel="stylesheet" href="http://www.google.com/cse/style/look/default.css" />
</head>
<body>
<div class="header">
        <h1>&amp;quot;<strong>www.123hjaf9hu32iufhuihoafine.com     </strong>&amp;quot; kann nicht gefunden
          ↪  werden.</h1>
        <hr style="display: none;" />
</div>
        <div id="cse" style="width: 95%;">Loading</div>
        <script src="http://www.google.com/jsapi" type="text/javascript"></script>
        <script type="text/javascript">
                var jsText = "www.123hjaf9hu32iufhuihoafine.com ";
                google.load('search', '1');
                google.setOnLoadCallback(function() {
                        var customSearchControl = new google.search.CustomSearchControl(
                          ↪  'partner-pub-5603398754456553:76a02bequj0');
                        var wsr  = new google.search.WebSearch();
                        customSearchControl.setResultSetSize(
                          ↪  google.search.Search.FILTERED_CSE_RESULTSET);
                        customSearchControl.draw('cse');
                        customSearchControl.addSearcher(wsr);
                        customSearchControl.execute(jsText);
                }, true);
        </script>
        <script type="text/javascript">
          var _gaq = _gaq || [];
          _gaq.push(['_setAccount', 'UA-809011-8']);
          _gaq.push(['_trackPageview']);
          (function() {
            var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async =
              ↪  true;
            ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') +
              ↪  '.google-analytics.com/ga.js';
            var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
          })();
        </script>
</body>
</html>
```

Listing 23: Returned Web page for a request of a non-existing domain using Drei

4.7 HTTP Caching and Manipulation (CM7)

In this test, the caching and manipulation of HTTP traffic was measured.

4.7.1 Caching

For all tested ISPs, there were no instances of caching. All facilitated repeated requests reached the server. This was true for all HTTP requests, not depending on the used

HTTP header or content type.

4.7.2 Manipulation

There were no differences in content for all tested ISPs. However, the HTTP header was manipulated for all measurements with T-Mobile.

T-Mobile kept adding a *Connection: keep-alive* header field and a *Host* header field to the transmitted HTTP header of both client and server. This did not alter the integrity of the body, which was transmitted unaltered in all tests. Listing 24 shows the sent and received HTTP alterations for T-Mobile for a sample measurement.

```
% Client request
GET /19cea6ff-5d64-4e47-af4e-62cdbd00574d/image2.jpg HTTP/1.1

% Client request, as received by server
GET /19cea6ff-5d64-4e47-af4e-62cdbd00574d/image2.jpg HTTP/1.1
Host: 128.130.204.34
Connection: keep-alive

% Server response
HTTP/1.1 200 OK
Content-Length: 15717310
Content-Language: en
Content-Type: image/jpeg

[JPG data]

% Server response, as received by the client
HTTP/1.1 200 OK
Content-Length: 15717310
Content-Language: en
Content-Type: image/jpeg
Connection: keep-alive

[JPG data]
```

Listing 24: HTTP communication for T-Mobile, showing added *Connection: keep-alive* and *Host*-headers

4.8 HTTP Antivirus test (VS7)

In this measurement, the transmission of the EICAR-test file was tested. The file was transferred without alterations for all ISPs, meaning that no anti-virus-inspection is

taking place in the concrete test setup. However, the HTTP header was altered for T-Mobile customers, as in the CM7 measurement, but the transmitted file was not affected by this.

4.9 Invalid HTTP syntax test (HTTP7)

Malformed HTTP requests were transmitted without alterations for all ISPs with the exception of T-Mobile. T-Mobile once again added the *Connection: Keep-alive* and *Host* headers. Additionally, a syntax error was corrected by T-Mobiles middlebox. The traffic can be seen in Listing 25

```
1  % Client request
2  GET /19cea6ff-5d64-4e47-af4e-62cdbd00574d/faultyResponse
    ↪ HTTP/1.1
3
4  % Client request, as  received by server
5  GET /19cea6ff-5d64-4e47-af4e-62cdbd00574d/faultyResponse
    ↪ HTTP/1.1
6  Host: 128.130.204.34
7  Connection: keep-alive
8
9  % Server response
10 HTTP/79.2 404 OK
11 Content-Length: 11
12 Content-Language: en
13 Content-Type: text/plain
14
15 testfile123
16
17 % Server response, as received by the client
18 HTTP/1.1 404 Not Found
19 Content-Length: 11
20 Content-Language: en
21 Content-Type: text/plain
22 Connection: keep-alive
```

Listing 25: HTTP communication for T-Mobile, showing added *Connection: keep-alive* and *Host*-headers and a corrected HTTP/79.2

	jitter (ms)	jitter (ms) max
A1 DSL	40,82	55,41
A1 LTE	39,42	47,85
Drei	43,91	60,52
TMA	49,74	61,94
UPC	22,96	43,64
Average	**42,33**	**55,54**

Table 4.6: Jitter for tested VOIP calls

4.10 Voice over IP test (VOIP7)

When testing Voice over IP connection quality, none of the providers had significantly worse connection quality than the others. In Table 4.6, the mean jitter, and the mean maximal jitter are listed. The best results were obtained with UPC, but the results of the other ISPs were not significantly worse.

There were no significant variations in connection quality over the course of the day. However, it should be noted that due to Firewall problems, the sample size of $n=275$ was not large enough to definitively conclude connection variations over the course of a day.

4.11 Malformed TLS handshake test (TLS4)

All TLS handshakes, regardless of syntax errors, were received by both client and server without interference for all tested ISPs.

4.12 Invalid POP3 syntax test (POP37)

All POP3 communication, regardless of syntax errors, was received by both client and server without interference and manipulation for all tested ISPs.

4.13 Invalid SMTP syntax text (SMTP7)

All SMTP communication, regardless of syntax errors, was received by both client and server without interference and manipulation for all tested ISPs.

Note: Due to the Firewall configuration used in the infrastructure provided by the university, port 25 could not be tested. Instead, all communication was done on port 8025.

4.14 StartTLS stripping test (STLS7)

All SMTP communication was received by both client and server without interference and manipulation for all tested ISPs. There was no case of STARTTLS-stripping measured.

	Down 80	Up 80	Down 443	Up 443	Down 6881	Up 6881	Down 48123	Up 48123
A1 DSL	2,61	0,69	2,54	0,69	2,50	0,69	2,44	0,68
A1 LTE	5,61	2,43	5,50	2,49	5,62	2,45	5,61	2,45
Drei	7,26	3,33	7,15	3,38	7,23	3,32	7,16	3,35
TMA	4,89	2,17	4,89	2,14	4,96	2,12	4,89	1,98
UPC	8,98	1,25	8,86	1,26	8,89	1,25	8,73	1,20
Average	**5,42**	**2,04**	**5,35**	**2,06**	**5,40**	**2,04**	**5,34**	**2,00**

Table 4.7: Test speeds (Mbps) on different ports for the tested ISPs

Note: Due to the Firewall configuration used in the infrastructure provided by the university, port 25 could not be tested. Instead, all communication was done on port 8025.

4.15 TCP bandwidth test (TCPS4)

During the measurement period, every hour, a speed test was conducted. The transmission speed was tested on ports 80 (HTTP), 443 (HTTPS), 6881 (BitTorrent) and 48123 (control port).

Due to an error in the measurement, speed tests in the time period from 2016-08-04 to 2016-09-21 were showing slower connection speeds than the ISPs provided in reality. This can be seen in Figure 4.7. To remedy this, the tests during this period have been discarded and additional tests conducted before the start and after the end of the measurement period (using the same test programming) have been added. The unaltered connection speeds can be seen in Table 4.7, adjusting for the invalid measurements in Table 4.8.

4.15.1 Speed differences depending on port

For all tested ISPs, there were no overall differences in connection speeds depending on the used port. If traffic shaping does take place, it could not be measured overall.

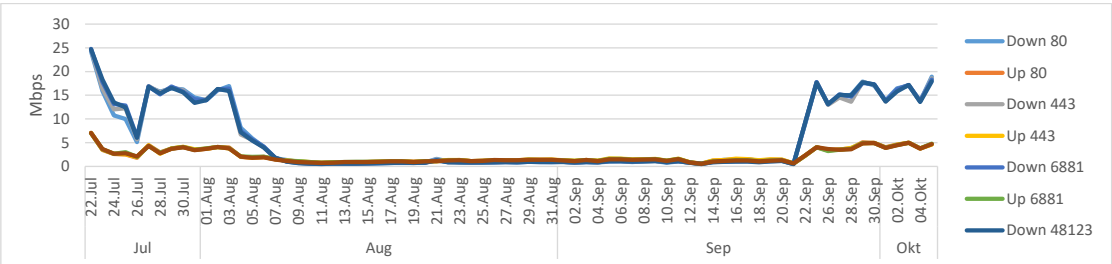

Figure 4.5: Measured connection speeds during the measurement period

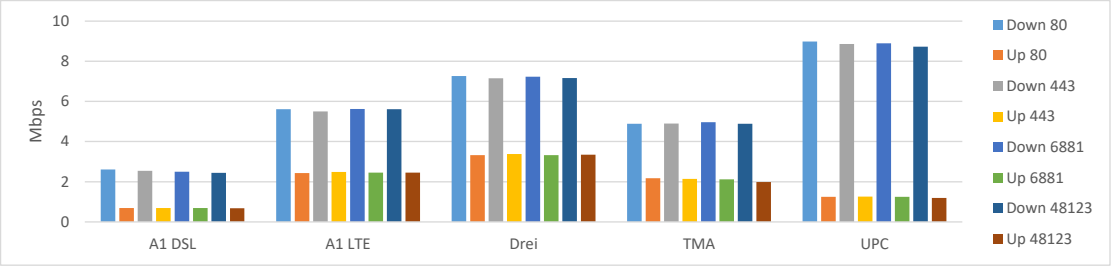

Figure 4.6: Test speeds (Mbps) on different ports for the tested ISPs, corresponding to Table 4.7

	Down 80	Up 80	Down 443	Up 443	Down 6881	Up 6881	Down 48123	Up 48123
A1 DSL	5,90	0,70	5,91	0,70	5,92	0,70	5,82	0,69
A1 LTE	16,99	5,18	16,83	5,20	17,20	5,24	17,18	5,19
Drei	22,17	7,94	22,01	7,94	22,21	7,93	22,11	7,91
TMA	13,38	3,87	13,56	3,84	13,72	3,82	13,56	3,77
UPC	18,11	1,36	18,24	1,38	18,33	1,37	18,20	1,36
Average	**14,72**	**3,84**	**14,71**	**3,84**	**14,88**	**3,85**	**14,78**	**3,82**

Table 4.8: Test speeds (Mbps) on different ports for the tested ISPs, adjusted

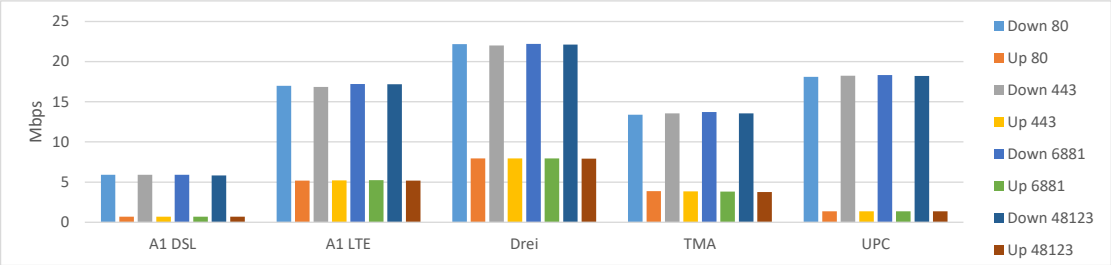

Figure 4.7: Test speeds (Mbps) on different ports for the tested ISPs, corresponding to Table 4.8

4.15.2 Speed differences depending on the time of the day

All tests were executed once per hour. The resulting speed measurements can be seen in Tables 4.9 and 4.10, giving the connection speed depending on the hour of the day in the UTC timezone. The connection speeds are visualized in Figure 4.8 for A1 LTE, Figure 4.9 for A1 DSL, Figure 4.10 for UPC, Figure 4.11 for Drei and Figure 4.12 for T-Mobile Austria.

All ISPs seemed to yield slower connection speeds during the night. However, it should be noted that throughout the day, the connection speeds stays approximately the same for all used ports. No traffic shaping could be measured.

Hour of the day	Provider	Down 80	Up 80	Down 443	Up 443	Down 6881	Up 6881	Down 48123	Up 48123
0	**Overall**	**15,40**	**3,98**	**15,44**	**3,94**	**15,94**	**4,00**	**15,44**	**3,93**
	A1 LTE	17,00	4,95	17,01	4,82	16,97	4,82	16,37	4,74
	A1 DSL	5,65	0,72	6,11	0,71	6,31	0,71	5,62	0,70
	UPC	21,79	1,47	22,08	1,47	22,62	1,47	22,63	1,47
	Drei	24,33	8,83	23,56	8,82	24,15	8,83	22,87	8,70
	TMA	13,83	3,88	13,98	3,84	15,13	4,05	15,30	3,97
1	**Overall**	**10,12**	**3,48**	**9,57**	**3,29**	**9,53**	**3,35**	**9,27**	**3,11**
	A1 LTE	11,24	4,33	10,23	4,06	10,05	4,14	10,14	3,81
	A1 DSL	4,82	0,71	4,97	0,72	5,05	0,71	4,65	0,72
	UPC	10,19	1,23	9,34	1,23	8,53	1,23	7,51	1,19
	Drei	16,17	7,47	15,12	6,93	14,81	7,07	15,16	6,57
	TMA	9,71	3,61	9,40	3,48	9,97	3,52	9,49	3,24
2	**Overall**	**14,73**	**3,69**	**15,09**	**3,69**	**15,05**	**3,70**	**14,49**	**3,65**
	A1 LTE	16,74	4,99	16,77	5,03	16,79	5,00	16,84	4,94
	A1 DSL	5,88	0,69	6,09	0,70	5,85	0,70	5,53	0,69
	UPC	18,08	1,26	19,39	1,33	19,32	1,34	17,66	1,35
	Drei	22,59	8,09	23,63	8,06	23,55	8,11	23,54	8,02
	TMA	13,67	3,71	13,33	3,67	13,47	3,68	12,47	3,60
3	**Overall**	**14,13**	**3,65**	**14,35**	**3,58**	**14,59**	**3,71**	**14,25**	**3,65**
	A1 LTE	16,82	4,99	16,95	4,88	17,24	5,06	17,06	4,95
	A1 DSL	5,78	0,72	5,33	0,71	5,53	0,71	5,55	0,71
	UPC	14,14	1,17	15,49	1,27	15,66	1,27	14,84	1,24
	Drei	22,62	7,96	23,13	7,70	23,52	8,15	23,14	8,00
	TMA	13,64	3,70	13,67	3,63	13,82	3,66	13,36	3,65
4	**Overall**	**8,34**	**2,65**	**8,53**	**2,53**	**8,50**	**2,61**	**8,52**	**2,55**
	A1 LTE	8,75	2,91	8,61	2,88	9,03	2,83	8,74	2,82
	A1 DSL	3,85	0,70	3,93	0,69	4,15	0,70	4,33	0,68
	UPC	11,34	1,14	12,95	1,24	13,20	1,22	13,70	1,29
	Drei	12,04	5,33	11,78	5,05	11,61	5,36	11,61	5,16
	TMA	8,32	2,89	8,66	2,64	7,98	2,77	7,93	2,68
5	**Overall**	**14,98**	**4,02**	**14,84**	**3,83**	**14,03**	**3,76**	**14,40**	**3,71**
	A1 LTE	17,24	4,93	16,91	4,87	15,90	4,84	16,79	4,79
	A1 DSL	5,85	0,72	5,74	0,72	5,30	0,70	5,41	0,71
	UPC	18,72	1,41	19,00	1,46	17,84	1,45	17,20	1,42
	Drei	22,98	8,58	22,80	8,00	21,81	7,82	21,98	7,67
	TMA	13,30	3,95	13,12	3,66	12,44	3,58	13,30	3,55
6	**Overall**	**12,83**	**3,49**	**12,82**	**3,44**	**12,32**	**3,41**	**12,49**	**3,37**
	A1 LTE	15,48	4,58	14,95	4,61	14,70	4,59	14,54	4,54
	A1 DSL	5,14	0,66	5,06	0,67	4,89	0,67	4,83	0,66
	UPC	15,68	1,31	16,23	1,32	14,91	1,31	15,14	1,29
	Drei	19,62	7,54	19,28	7,36	18,65	7,25	18,85	7,28
	TMA	11,15	3,58	11,47	3,49	11,07	3,48	11,66	3,35
7	**Overall**	**14,47**	**3,93**	**14,66**	**3,93**	**14,70**	**3,99**	**14,67**	**4,01**
	A1 LTE	17,14	5,05	16,92	4,95	16,97	5,04	17,12	5,21
	A1 DSL	5,55	0,69	5,62	0,69	5,62	0,70	5,74	0,69
	UPC	19,24	1,34	19,72	1,36	19,01	1,34	19,02	1,33
	Drei	22,59	8,19	23,05	8,05	23,14	8,46	23,04	8,44
	TMA	12,94	3,86	13,29	4,06	13,56	3,92	13,28	3,87
8	**Overall**	**11,63**	**2,97**	**11,66**	**3,01**	**11,64**	**3,02**	**11,82**	**3,02**
	A1 LTE	12,78	3,98	12,77	4,05	12,70	4,03	12,95	4,10
	A1 DSL	5,14	0,64	5,04	0,64	5,05	0,65	5,09	0,64
	UPC	17,85	1,26	17,80	1,27	17,54	1,26	17,88	1,24
	Drei	16,91	6,33	17,09	6,46	17,30	6,51	17,56	6,54
	TMA	9,36	2,92	9,51	2,91	9,47	2,91	9,56	2,87
9	**Overall**	**14,42**	**3,76**	**14,07**	**3,65**	**14,55**	**3,57**	**14,27**	**3,53**
	A1 LTE	16,74	5,12	16,41	4,93	17,04	4,81	16,81	4,77
	A1 DSL	5,91	0,69	5,64	0,69	5,58	0,69	5,55	0,68
	UPC	18,30	1,32	17,34	1,38	17,57	1,35	17,94	1,32
	Drei	23,23	8,37	23,05	8,02	23,79	7,80	23,14	7,75
	TMA	12,83	3,66	12,60	3,60	13,44	3,55	12,79	3,46
10	**Overall**	**15,16**	**4,05**	**15,15**	**4,17**	**14,96**	**4,16**	**15,00**	**4,14**
	A1 LTE	17,04	5,41	17,28	5,60	17,26	5,37	17,00	5,54
	A1 DSL	5,96	0,71	6,03	0,70	5,94	0,70	5,81	0,71
	UPC	19,66	1,45	19,76	1,47	19,57	1,46	19,60	1,43
	Drei	23,54	8,47	22,46	8,71	22,00	9,12	22,05	8,95
	TMA	13,10	3,87	13,58	3,98	13,39	3,84	13,88	3,76
11	**Overall**	**17,34**	**4,28**	**17,63**	**4,25**	**17,65**	**4,25**	**17,59**	**4,36**
	A1 LTE	19,79	5,80	20,18	5,85	20,15	5,83	20,36	5,80
	A1 DSL	6,59	0,72	6,54	0,71	6,65	0,72	6,59	0,72
	UPC	22,12	1,46	22,05	1,47	21,89	1,46	21,41	1,46
	Drei	26,28	8,52	26,81	8,48	27,05	8,44	27,01	8,85
	TMA	15,00	4,14	15,50	4,04	15,41	4,07	15,31	4,23

Table 4.9: Speed measurements depending on the hour of the day, 0-11 UTC

Hour of the day	Provider	Down 80	Up 80	Down 443	Up 443	Down 6881	Up 6881	Down 48123	Up 48123
12	**Overall**	**17,19**	**4,26**	**17,26**	**4,54**	**17,44**	**4,30**	**17,50**	**4,40**
	A1 LTE	21,39	6,23	21,18	6,60	21,56	6,24	21,91	6,25
	A1 DSL	6,61	0,70	6,45	0,70	6,26	0,70	6,44	0,70
	UPC	19,60	1,40	18,97	1,46	18,84	1,45	19,05	1,42
	Drei	24,85	8,88	25,67	9,69	26,45	8,98	26,01	9,41
	TMA	16,32	4,34	16,67	4,52	16,77	4,37	16,83	4,45
13	**Overall**	**15,29**	**3,89**	**15,26**	**3,94**	**15,64**	**3,91**	**15,41**	**3,89**
	A1 LTE	17,75	5,45	18,26	5,58	18,56	5,60	18,18	5,56
	A1 DSL	6,60	0,71	6,51	0,71	6,60	0,71	6,44	0,70
	UPC	18,60	1,34	17,68	1,34	18,91	1,34	18,47	1,34
	Drei	21,05	7,75	20,78	7,82	21,10	7,72	20,82	7,73
	TMA	14,14	4,05	14,45	4,10	14,67	4,00	14,72	3,97
14	**Overall**	**15,52**	**3,95**	**15,05**	**4,08**	**15,50**	**3,95**	**15,40**	**3,96**
	A1 LTE	17,75	5,35	16,98	5,56	18,22	5,37	17,69	5,35
	A1 DSL	6,21	0,69	6,03	0,69	5,96	0,69	6,08	0,70
	UPC	18,23	1,31	18,61	1,29	19,15	1,31	19,48	1,32
	Drei	22,79	7,70	21,40	8,11	21,94	7,68	22,75	7,75
	TMA	14,73	3,86	14,60	3,87	14,67	3,85	13,77	3,86
15	**Overall**	**17,51**	**4,49**	**17,66**	**4,55**	**17,76**	**4,59**	**17,82**	**4,56**
	A1 LTE	19,96	6,02	19,81	6,08	20,31	6,04	20,35	6,04
	A1 DSL	7,00	0,70	7,02	0,69	6,82	0,69	6,78	0,69
	UPC	21,70	1,47	22,03	1,46	22,44	1,48	22,49	1,47
	Drei	26,28	8,66	26,49	9,01	25,87	9,22	26,39	9,08
	TMA	16,37	4,63	16,77	4,57	17,28	4,58	17,06	4,55
16	**Overall**	**13,08**	**3,79**	**13,16**	**3,77**	**13,53**	**3,77**	**12,99**	**3,76**
	A1 LTE	17,84	5,49	17,36	5,62	17,12	5,57	16,45	5,58
	A1 DSL	5,85	0,70	5,83	0,70	5,90	0,70	5,86	0,69
	UPC	13,55	1,36	13,78	1,38	15,31	1,35	14,13	1,35
	Drei	18,21	7,03	18,33	6,89	18,56	6,96	17,72	6,96
	TMA	11,07	3,78	11,60	3,69	12,31	3,70	11,98	3,63
17	**Overall**	**16,88**	**4,53**	**17,06**	**4,53**	**17,20**	**4,56**	**17,31**	**4,55**
	A1 LTE	17,49	6,17	17,63	6,18	17,75	6,32	18,60	6,29
	A1 DSL	6,93	0,72	7,23	0,72	7,32	0,72	7,05	0,72
	UPC	20,93	1,43	20,40	1,42	20,10	1,41	21,19	1,41
	Drei	27,32	9,98	27,59	9,94	28,11	9,95	27,94	9,87
	TMA	14,72	4,49	15,19	4,54	15,42	4,54	14,79	4,55
18	**Overall**	**13,89**	**3,84**	**13,90**	**3,81**	**14,22**	**3,99**	**13,93**	**3,92**
	A1 LTE	16,31	5,22	16,46	5,25	16,93	5,70	17,69	5,54
	A1 DSL	5,34	0,70	5,66	0,70	5,79	0,70	5,45	0,69
	UPC	15,93	1,38	15,31	1,37	15,79	1,40	15,05	1,39
	Drei	20,47	7,33	20,65	7,39	20,54	7,67	20,65	7,67
	TMA	12,89	4,00	12,71	3,83	13,35	3,94	12,01	3,80
19	**Overall**	**15,24**	**3,70**	**15,30**	**3,78**	**16,28**	**3,77**	**16,01**	**3,80**
	A1 LTE	17,37	5,06	17,46	5,19	19,93	5,42	19,14	5,48
	A1 DSL	5,82	0,71	5,59	0,72	6,03	0,71	6,03	0,71
	UPC	19,47	1,40	20,57	1,42	20,26	1,40	20,04	1,40
	Drei	22,61	7,35	21,20	7,52	23,11	7,27	22,79	7,31
	TMA	13,72	3,79	14,55	3,85	14,73	3,84	14,70	3,86
20	**Overall**	**13,46**	**3,35**	**13,62**	**3,32**	**13,57**	**3,32**	**13,80**	**3,29**
	A1 LTE	14,83	4,45	14,39	4,39	14,94	4,72	16,21	4,59
	A1 DSL	5,81	0,70	5,70	0,69	5,57	0,68	5,25	0,67
	UPC	15,27	1,40	16,03	1,37	16,47	1,35	15,91	1,32
	Drei	20,56	6,58	19,85	6,51	19,86	6,37	20,13	6,34
	TMA	12,59	3,55	13,89	3,58	13,01	3,43	13,31	3,45
21	**Overall**	**18,40**	**4,26**	**17,91**	**4,29**	**18,63**	**4,24**	**18,41**	**4,17**
	A1 LTE	21,84	6,03	21,67	6,09	22,99	6,22	22,31	6,10
	A1 DSL	6,89	0,73	6,49	0,72	6,44	0,72	6,18	0,72
	UPC	22,69	1,48	22,61	1,48	23,24	1,48	22,58	1,48
	Drei	26,89	8,54	26,45	8,75	28,06	8,56	27,98	8,49
	TMA	17,17	4,45	16,01	4,34	16,31	4,19	16,66	4,05
22	**Overall**	**16,91**	**4,47**	**17,14**	**4,45**	**17,50**	**4,50**	**17,46**	**4,38**
	A1 LTE	19,90	6,30	19,62	6,14	19,08	6,28	19,13	5,96
	A1 DSL	6,24	0,72	6,61	0,71	6,82	0,71	6,99	0,69
	UPC	22,15	1,47	22,31	1,47	22,75	1,47	22,79	1,48
	Drei	24,33	9,38	24,62	9,39	25,05	9,39	24,47	9,36
	TMA	15,35	4,33	15,87	4,38	17,09	4,48	17,15	4,28
23	**Overall**	**14,98**	**3,59**	**14,76**	**3,57**	**14,91**	**3,64**	**14,97**	**3,59**
	A1 LTE	16,54	4,68	15,81	4,70	17,49	5,04	16,83	4,77
	A1 DSL	6,07	0,70	6,42	0,69	6,52	0,69	6,18	0,69
	UPC	17,95	1,31	17,60	1,31	17,74	1,31	19,89	1,31
	Drei	22,16	7,46	21,99	7,47	21,47	7,49	21,30	7,59
	TMA	14,65	3,74	14,28	3,65	13,76	3,63	13,70	3,58

Table 4.10: Speed measurements depending on the hour of the day, 12-24 UTC

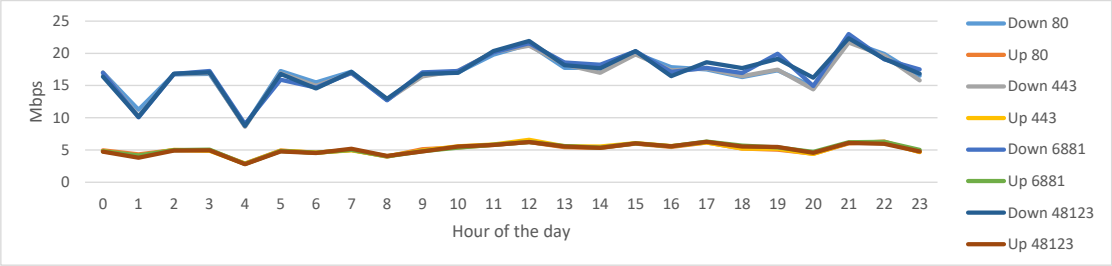

Figure 4.8: Test speed on different ports depending on the UTC hour of day for A1 LTE

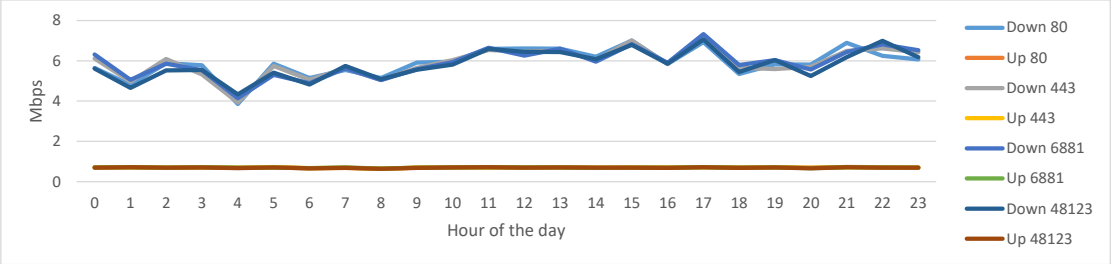

Figure 4.9: Test speeds on different ports depending on the UTC hour of day for A1 DSL

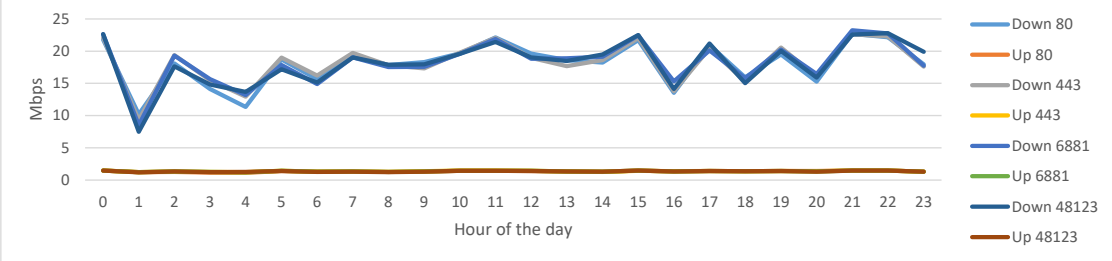

Figure 4.10: Test speeds on different ports depending on the UTC hour of day for UPC

4.16 Multimedia test (MM7)

For the multimedia test, the speed test was initiated on port 80 with an HTTP header posing as a YouTube-video. The motivation behind this was to detect if ISPs discriminate based on this header field. For establishing a reference value, the same test was conducted on port 48123.

The measurement had the same problems as the TCPS4 test, therefore tests during the period of 2016-08-04 to 2016-09-21 were discarded.

During development, another problem occurred: For tests with T-Mobile, the down-

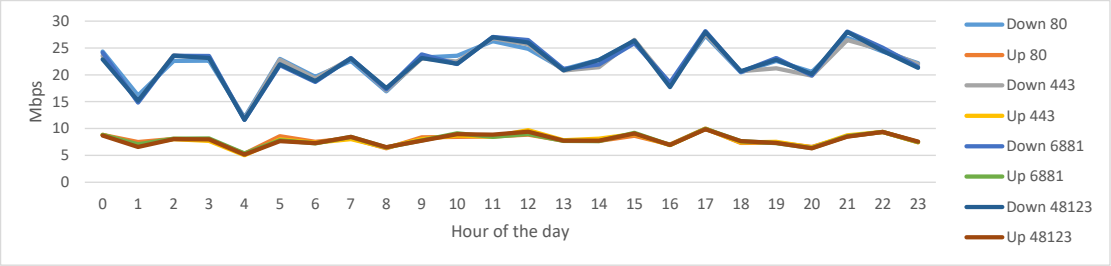

Figure 4.11: Test speeds on different ports depending on the UTC hour of day for Drei

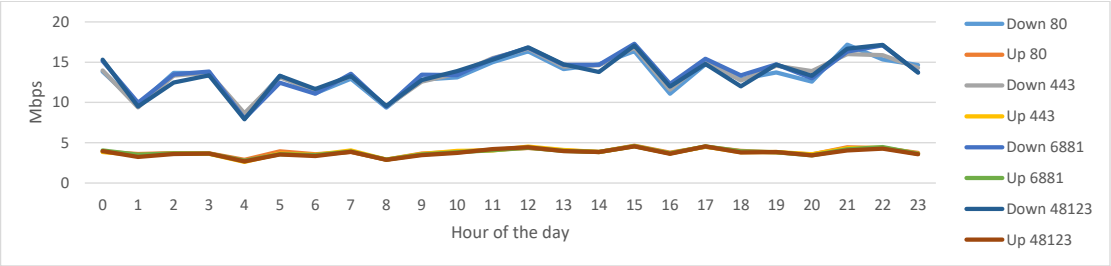

Figure 4.12: Test speeds on different ports depending on the UTC hour of day for T-Mobile

load test phase sometimes did not finish correctly. This was attributed to the middlebox used by T-Mobile for HTTP traffic on Port 80: Only traffic volume less or equal to the *Content-Length*-field in the HTTP header is forwarded to the communication partner. This problem was mitigated by using a *Content-Length* large enough to allow for all connection speeds. Due to the middlebox, performing an upload speed test was also not possible, since all segments sent over the same TCP connection that was used for the download test did never reach the communication partner but were filtered out by T-Mobile.

4.16.1 Speed differences depending on the HTTP header

As can be seen in Table 4.12 and Figure 4.13, no significant differences could be measured depending on HTTP headers and port. If an ISP behavior similar to T-Mobile USA's *BingeOn* was expected, this should have lead to significant lower connection speeds on TCP port 80. Instead, connection speeds for the tested ISPs seem to be consistent for the tested HTTP headers, and also consistent with the non-HTTP speed measurements conducted in measurement TCPS4.

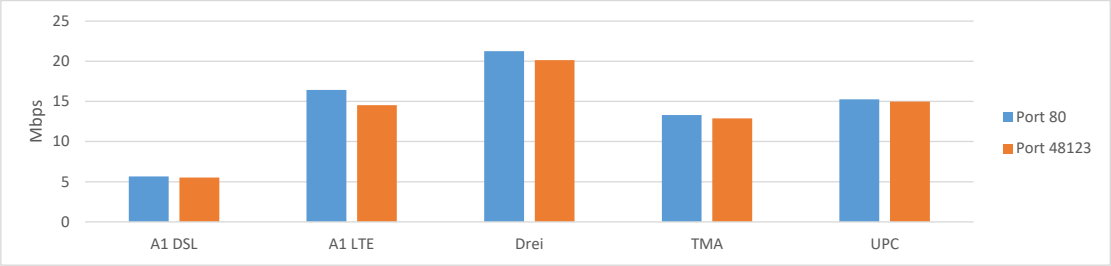

Figure 4.13: Connection speeds depending on port and HTTP header (Table 4.11)

	Port 80	**Port 48123**
A1 DSL	5,66	5,52
A1 LTE	16,44	14,53
Drei	21,25	20,13
TMA	13,31	12,87
UPC	15,25	14,98
Average	**14,03**	**13,26**

Table 4.11: Connection speeds (Mbps) depending on port and HTTP header

4.16.2 Speed differences depending on the time of the day

Similar to TCPS4, variations in connection speed during the day were measured. As can be seen in Table 4.12, no significant differences could be measured depending on the HTTP header and port. Traffic shaping based on the HTTP header was not detected for any of the tested providers.

4.17 Traceroute test (TRAC3)

In the traceroute measurement, a traceroute was periodically performed for all providers. For this, the hosts *www.google.at*, *www.derstandard.at* and *www.orf.at* were used. These sites can be found on rank 1, 7 and 10 of the Alexa Top 100 Websites visited in Austria[2] and could therefore be a prime target for optimization by ISPs.

The results of the traceroute measurements are given in Tables 4.13, 4.14, 4.15, 4.16 and 4.17. In these tables, the last IP of each traceroute is given, along with the number of measurements that had this concrete IP as their last entry and the mean TTL value that was used to reach the IP.

Accordingly to these tables, *DerStandard.at* uses a single IP, while *Google* and *ORF.at* use a load-balancing system consisting of multiple IP addresses. Even for *DerStandard.at*, not every traceroute resulted in the correct IP. Some measurements returned IPs belonging to the ISP subnet or network infrastructure. The cause for this could e.g. be network issues, downtimes of the service or changing firewall configuration.

Hour of the day	Port	A1 DSL	A1 LTE	Drei	TMA	UPC	Overall
0	Port 80	5,58	16,27	22,60	14,34	13,92	**14,14**
	Port 48123	5,53	14,28	21,90	13,95	13,01	**13,40**
	# tests	18	13	15	14	14	**74**
1	Port 80	5,98	18,53	23,57	15,66	15,80	**15,45**
	Port 48123	5,73	15,75	22,06	14,67	15,74	**14,39**
	# tests	18	13	15	16	14	**76**
2	Port 80	4,68	13,54	17,88	11,44	11,78	**11,55**
	Port 48123	4,47	12,09	15,98	10,90	10,53	**10,53**
	# tests	18	14	15	15	12	**74**
3	Port 80	6,18	18,58	24,45	15,69	15,80	**15,68**
	Port 48123	6,04	16,85	23,71	15,62	15,69	**15,17**
	# tests	18	13	15	16	14	**76**
4	Port 80	5,11	17,00	20,79	12,76	15,19	**13,63**
	Port 48123	5,09	15,46	20,80	12,91	14,81	**13,31**
	# tests	19	14	15	15	13	**76**
5	Port 80	4,53	12,47	16,54	10,05	9,30	**10,25**
	Port 48123	3,94	10,62	14,07	9,81	8,71	**9,14**
	# tests	18	13	14	15	11	**71**
6	Port 80	5,83	19,63	20,83	13,17	11,68	**13,90**
	Port 48123	5,30	17,02	18,61	12,36	10,09	**12,43**
	# tests	17	12	15	15	11	**70**
7	Port 80	4,63	14,28	19,90	11,51	17,26	**13,07**
	Port 48123	4,73	12,39	18,63	11,17	16,76	**12,31**
	# tests	16	14	14	15	11	**70**
8	Port 80	6,02	16,95	21,94	12,70	15,46	**14,31**
	Port 48123	5,61	14,51	19,48	11,73	16,23	**13,20**
	# tests	17	16	14	15	14	**76**
9	Port 80	6,12	17,14	24,81	15,06	17,34	**15,48**
	Port 48123	6,08	15,52	25,23	14,52	19,17	**15,50**
	# tests	19	15	14	13	15	**76**
10	Port 80	6,68	19,85	26,03	13,84	16,79	**16,14**
	Port 48123	6,49	18,22	24,24	13,17	16,79	**15,27**
	# tests	18	16	14	15	11	**74**
11	Port 80	6,55	20,23	26,81	15,35	20,30	**17,27**
	Port 48123	6,47	17,77	27,45	15,23	19,80	**16,74**
	# tests	19	17	15	17	14	**82**
12	Port 80	7,11	18,37	24,37	14,54	19,87	**16,48**
	Port 48123	6,80	15,43	22,96	14,37	18,49	**15,30**
	# tests	18	15	16	17	14	**80**
13	Port 80	5,92	16,89	18,63	14,14	17,40	**14,41**
	Port 48123	5,93	15,61	18,13	13,79	17,12	**13,92**
	# tests	18	18	16	17	15	**84**
14	Port 80	6,62	17,11	21,39	14,46	14,84	**14,71**
	Port 48123	6,79	15,27	20,61	13,82	14,57	**14,03**
	# tests	18	17	16	17	14	**82**
15	Port 80	6,69	18,73	23,15	14,32	16,65	**15,65**
	Port 48123	6,36	16,32	20,46	13,74	16,84	**14,47**
	# tests	18	16	16	17	14	**81**
16	Port 80	4,87	13,27	17,10	10,41	11,86	**11,21**
	Port 48123	5,13	12,32	16,54	10,57	12,22	**11,07**
	# tests	19	17	15	17	16	**84**
17	Port 80	5,03	14,35	19,91	12,79	16,87	**13,50**
	Port 48123	5,01	13,30	18,50	12,04	16,17	**12,72**
	# tests	18	17	16	17	14	**82**
18	Port 80	4,52	13,56	15,72	11,21	8,74	**10,68**
	Port 48123	4,32	11,65	14,99	10,54	9,03	**10,02**
	# tests	19	18	17	17	15	**86**
19	Port 80	6,15	16,91	24,61	15,06	17,90	**15,71**
	Port 48123	5,72	13,74	20,65	13,94	17,11	**13,83**
	# tests	20	17	17	17	14	**85**
20	Port 80	4,89	12,61	19,43	11,19	18,98	**12,83**
	Port 48123	4,90	11,66	19,95	11,12	19,13	**12,75**
	# tests	18	16	15	16	12	**77**
21	Port 80	5,49	17,08	19,47	14,15	14,17	**13,70**
	Port 48123	5,37	15,08	19,03	13,52	13,24	**12,92**
	# tests	19	15	16	16	12	**78**
22	Port 80	6,09	19,35	25,66	15,38	16,08	**16,14**
	Port 48123	5,82	17,44	25,33	15,26	15,95	**15,59**
	# tests	19	16	16	16	14	**81**
23	Port 80	4,59	12,09	15,15	9,79	10,96	**10,31**
	Port 48123	4,62	10,35	14,70	9,78	10,49	**9,82**
	# tests	18	14	16	15	13	**76**
Overall	**Port 80**	**5,66**	**16,44**	**21,25**	**13,31**	**15,25**	**14,03**
	Port 48123	**5,52**	**14,53**	**20,13**	**12,87**	**14,98**	**13,26**
	# tests	**437**	**366**	**367**	**380**	**321**	**1871**

Table 4.12: Speed measurements depending on the hour of the day UTC

A1 LTE		
www.derstandard.at	**www.google.at**	**www.orf.at**
194.116.243.20 (1521, Ø 5,2)	216.58.209.67 (435, Ø 4,28)	194.232.104.149 (371, Ø 4,71)
10.122.6.178 (35, Ø 25,83)	216.58.214.195 (250, Ø 4,75)	194.232.104.142 (322, Ø 5,52)
195.3.64.142 (21, Ø 29,29)	216.58.214.227 (92, Ø 6,43)	194.232.104.150 (321, Ø 4,62)
195.3.90.33 (11, Ø 29,91)	216.58.212.67 (81, Ø 3)	194.232.104.140 (315, Ø 5,09)
10.122.6.161 (11, Ø 26)	172.217.23.99 (81, Ø 3)	194.232.104.139 (212, Ø 5,6)
	172.217.16.99 (62, Ø 4,9)	194.232.104.141 (166, Ø 4,61)
	216.58.208.163 (57, Ø 3)	10.122.6.178 (70, Ø 23,87)
	216.58.212.163 (56, Ø 4)	10.122.6.161 (34, Ø 23,94)
	64.233.162.94 (50, Ø 3,78)	194.158.150.61 (21, Ø 29)
	173.194.73.94 (46, Ø 4,65)	10.122.5.73 (18, Ø 29)

Table 4.13: Last IP with measurement count and mean TTL values for A1 LTE

The mean traceroutes differ between ISPs, as shown in Figure 4.14. TTL values for A1 LTE are generally the lowest, ranging from 5 to 15 and the highest for Drei, ranging from 21 to 25.

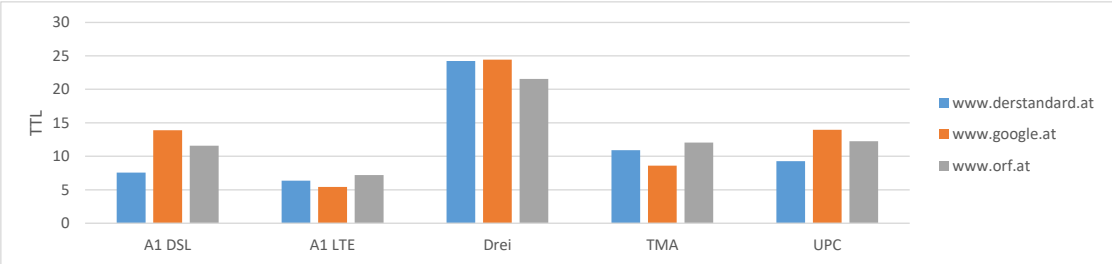

Figure 4.14: Mean TTL values for the traceroutes performed

Once again, there could be no significant difference in TTL be measured depending on the time of the day. This can be seen in Figure 4.15 and Table 4.18. For these results, the TTL values for all traceroutes performed to *www.derstandard.at* resulting in IP *194.116.243.20* as their last entries were used.

It can be concluded that none of the tested ISPs in the used test setup cached the website on the ISP infrastructure during certain times of the day. If this would be the case, the number of tests resolving to the IP belonging to *DerStandard.at* should not be consistent throughout the day. Since even the TTL values are consistent throughout the day, none of the tested ISPs seems to be optimizing routing during times of heavy load, resulting in roughly the same route for all times of the day.

A1 DSL		
www.derstandard.at	**www.google.at**	**www.orf.at**
194.116.243.20 (1208, Ø 5,63)	216.58.213.99 (110, Ø 14,17)	194.232.104.150 (256, Ø 7,94)
195.3.68.133 (54, Ø 31)	216.58.214.3 (99, Ø 14,28)	194.232.104.140 (234, Ø 7,86)
195.3.64.142 (48, Ø 30,06)	216.58.210.35 (96, Ø 12,82)	194.232.104.141 (234, Ø 7,68)
	216.58.198.163 (93, Ø 13,78)	194.232.104.139 (221, Ø 8,33)
	216.58.213.67 (73, Ø 13,34)	194.232.104.142 (220, Ø 8,6)
	172.217.23.3 (62, Ø 13,37)	194.232.104.149 (213, Ø 8,28)
	195.3.68.133 (51, Ø 30,75)	194.158.150.61 (111, Ø 29,68)
	216.58.208.131 (45, Ø 12,67)	194.158.150.145 (66, Ø 29,68)
	172.217.23.35 (41, Ø 13,83)	195.3.68.133 (49, Ø 30,43)
	172.217.16.99 (38, Ø 6,82)	195.3.64.10 (35, Ø 30,23)

Table 4.14: Last IP with measurement count and mean TTL values for A1 DSL

Drei		
www.derstandard.at	**www.google.at**	**www.orf.at**
194.116.243.20 (918, Ø 21,15)	92.60.6.245 (104, Ø 28,25)	194.232.104.139 (182, Ø 17,23)
213.94.72.8 (190, Ø 30)	213.94.72.8 (67, Ø 27,67)	194.232.104.149 (158, Ø 15,08)
213.94.72.5 (115, Ø 29,99)	216.58.201.163 (58, Ø 18,74)	194.232.104.140 (143, Ø 16,96)
213.94.72.6 (51, Ø 30)	216.58.209.163 (57, Ø 20,37)	194.158.150.149 (135, Ø 29,99)
213.94.72.11 (37, Ø 30)	216.58.209.99 (53, Ø 23,23)	194.232.104.142 (132, Ø 16,81)
213.94.72.77 (28, Ø 29,96)	216.58.214.227 (48, Ø 19,52)	194.232.104.141 (130, Ø 16,14)
213.94.72.81 (25, Ø 30)	213.94.72.77 (29, Ø 28,38)	194.232.104.150 (122, Ø 17,79)
213.94.72.37 (22, Ø 30)	216.58.211.35 (28, Ø 20,61)	194.158.150.41 (107, Ø 30)
213.94.72.33 (18, Ø 30)	216.58.209.195 (28, Ø 18,11)	213.94.72.11 (84, Ø 30)
193.203.0.214 (6, Ø 30)	172.217.16.99 (28, Ø 13,21)	213.94.72.9 (67, Ø 30)

Table 4.15: Last IP with measurement count and mean TTL values for Drei

T-Mobile Austria		
www.derstandard.at	**www.google.at**	**www.orf.at**
194.116.243.20 (1508, Ø 9,02)	173.194.222.94 (90, Ø 3,71)	194.232.104.139 (318, Ø 6,54)
10.74.133.22 (43, Ø 28,7)	173.194.221.94 (69, Ø 3,68)	194.232.104.140 (281, Ø 7,43)
10.74.139.22 (25, Ø 28,08)	10.126.5.2 (61, Ø 28,1)	194.232.104.141 (274, Ø 8,73)
10.15.16.22 (19, Ø 28,11)	64.233.161.94 (47, Ø 11,32)	194.232.104.149 (230, Ø 7,53)
10.15.12.162 (17, Ø 28,12)	74.125.205.94 (46, Ø 3,11)	194.232.104.150 (217, Ø 11,11)
10.126.7.2 (15, Ø 28,73)	173.194.73.94 (45, Ø 10,11)	194.232.104.142 (138, Ø 8,73)
10.15.16.46 (12, Ø 28,67)	10.74.133.22 (35, Ø 28,6)	10.74.133.22 (84, Ø 28,51)
10.15.16.34 (11, Ø 28,18)	216.58.201.163 (33, Ø 13,52)	10.126.7.2 (51, Ø 28,65)
10.126.5.2 (10, Ø 27,5)	74.125.232.247 (31, Ø 4,58)	10.74.139.22 (45, Ø 28,44)
10.15.16.58 (9, Ø 27,78)	74.125.232.248 (31, Ø 6,61)	10.126.5.2 (37, Ø 28,22)

Table 4.16: Last IP with measurement count and mean TTL values for TMA

UPC		
www.derstandard.at	**www.google.at**	**www.orf.at**
194.116.243.20 (1045, Ø 8,92)	74.125.206.94 (450, Ø 11,16)	194.232.104.139 (173, Ø 9,44)
84.116.231.37 (9, Ø 31)	216.58.211.3 (213, Ø 13,77)	194.232.104.140 (167, Ø 9,77)
213.46.173.6 (8, Ø 30)	216.58.211.35 (64, Ø 13,72)	194.232.104.150 (159, Ø 10,03)
	172.217.23.35 (58, Ø 15,19)	194.232.104.142 (158, Ø 10,38)
	216.58.208.35 (50, Ø 10,82)	194.232.104.149 (152, Ø 11,25)
	172.217.23.3 (46, Ø 14,72)	194.232.104.141 (141, Ø 10,06)
	213.46.160.77 (19, Ø 25,74)	194.158.150.61 (50, Ø 30,06)
	84.116.231.37 (15, Ø 23,53)	194.158.150.145 (45, Ø 30,02)
	216.239.51.59 (12, Ø 29,92)	84.116.231.37 (10, Ø 31)
	172.217.16.163 (12, Ø 11,75)	213.46.173.6 (6, Ø 30)

Table 4.17: Last IP with measurement count and mean TTL values for UPC

	A1 DSL		A1 LTE		Drei		TMA		UPC	
Hour of the day	**TTL**	**Count**	**TTL**	**Count**	**TTL**	**Count**	**TTL**	**Count**	**TTL**	**Count**
0	7,25	53	4,79	61	20,74	34	7,94	65	8,84	43
1	7,61	54	5,55	64	20,38	34	8,37	57	8,50	42
2	5,71	51	5,50	62	20,48	27	9,86	64	9,58	43
3	6,34	50	5,52	62	19,94	34	9,31	58	8,51	43
4	6,65	43	5,57	61	22,90	40	9,86	64	8,17	41
5	5,23	48	5,60	58	21,47	36	8,37	62	9,29	42
6	5,27	45	4,56	61	22,39	38	9,34	64	10,32	41
7	5,68	50	5,46	65	21,47	38	9,79	61	9,22	41
8	4,86	51	5,17	64	21,11	38	10,26	58	8,36	42
9	5,16	49	5,65	62	22,30	40	9,56	59	8,14	43
10	5,90	51	4,83	65	20,82	38	10,79	62	9,73	41
11	5,10	52	5,25	64	21,45	44	8,55	64	8,58	45
12	4,73	56	5,60	63	20,88	42	9,00	66	8,93	45
13	5,24	54	4,71	65	22,22	36	8,60	62	9,38	47
14	5,19	52	4,60	63	20,00	44	8,90	63	8,28	46
15	4,20	46	5,99	68	22,18	40	8,85	65	8,48	46
16	5,86	50	4,67	66	21,10	42	8,37	65	8,11	45
17	5,04	48	4,56	63	22,33	40	8,91	64	9,67	46
18	4,85	48	5,63	67	20,90	41	9,97	67	8,37	46
19	5,16	50	5,65	69	20,45	40	8,94	64	9,80	44
20	5,73	51	5,02	63	21,64	39	8,70	66	10,02	44
21	6,21	53	4,89	65	19,53	36	8,31	65	8,86	42
22	6,02	52	4,70	63	20,82	38	7,70	61	9,23	44
23	5,92	51	5,32	57	19,74	39	8,42	62	7,84	43

Table 4.18: Measurements for *DerStandard.at* during the day (UTC)

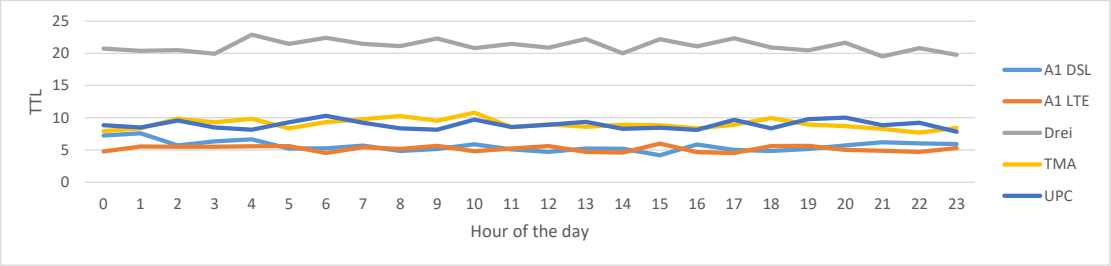

Figure 4.15: Mean TTL values for the traceroutes to *DerStandard.at* during the day (UTC)

4.18 Tor OONI test (OONI7)

The OONI7 measurement is mostly a supplementary test to the other metrics. It uses a well-established system and establishes a Tor connection for the control measurements. This Tor-connection was established for all Austrian ISPs. Due to the result structure produced by Tor OONI being in YAML, actually getting results in bulk required a custom Python script to convert relevant fields to JSON.

The OONI7 testdeck, as listed in Appendix B, contained three measurements, *manipulation/http_invalid_request_line*, *manipulation/http_header_field_manipulation* and *blocking/web_connectivity*. Based on this configuration, the framework measured HTTP header field tampering, tampering of invalid HTTP requests and the censorship of *kinox.to* and *movie4k.to*.

4.18.1 HTTP header tampering

Tor OONI detects four different forms of header manipulation:

- *Request Line Capitalization:* This measures, if there were any changes to the capitalization of the request line (e.g. *GET / HtTP/1.1*).
- *Header Field Number*: Measures, if the number of header fields matches the number of sent header fields
- *Header Name Capitalization:* The OONI probe sends the HTTP request using an unusual capitalization, e.g. *aCcEpt-chaRseT: 'ISO-8859-1,utf-8;q=0.7,*;q=0.3'*. This metric measures if the capitalization was changed mid-fly.
- *Header field value*: Detects any changes in the values of the sent fields.

Consistent with the results in metric *CM7*, none of the Austrian ISPs tampers with any of the HTTP header fields with the exception of T-Mobile. Interestingly, Tor OONI only detects a change in capitalization, but not the added *Connection: keep-alive* header. An example for such a request sent by Tor OONI can be seen in Listing 26.

```
% Client request heades
ACcEPT-LanGUAGE: 'en-US,en;q=0.8'
AccePT-eNCoDINg: 'gzip,deflate,sdch'
aCCEpt:
 ↪ 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
aCcEpt-chaRseT: 'ISO-8859-1,utf-8;q=0.7,*;q=0.3',
hOsT: 3nYjhGgxmtTJOie.com
uSer-agenT: 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
 ↪ rv:1.9.1.7) Gecko/20091221 Firefox/3.5.7'

% Request headers received by the Tor OONI server
Accept-Language: 'en-US,en;q=0.8'
Accept-Encoding: 'gzip,deflate,sdch'
Accept:
 ↪ 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
User-Agent: 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
 ↪ rv:1.9.1.7'Gecko/20091221 Firefox/3.5.7'
Accept-Charset: 'ISO-8859-1,utf-8;q=0.7,*;q=0.3'
Host: 3nyjhggxmttjoie.com
Connection: keep-alive
```

Listing 26: HTTP communication for T-Mobile, showing a added *Connection: keep-alive* HTTP header as well as changed capitalization and header order

4.18.2 HTTP invalid request line

In this test, Tor OONI sends invalid HTTP request lines and inspects, if they reach the server without interference. For this, different invalid requests are generated at random using four different methods.

- *Invalid method*: Here, an invalid HTTP method is generated at random, resulting e.g. in *RKDN / HTTP/1.1*
- *Invalid field count:* HTTP header fields are generated in a random (invalid) number, e.g. *N3YIr 8cAH7 CoB3C L39gK*
- *Big request method:* A random request method with 1024 bytes is generated, e.g. *1cd...lRKz / HTTP/1.1*
- *Invalid version number:* An invalid HTTP version number is generated and sent, e.g. *GET / HTTP/mzq*

When inspecting the results given in Table 4.19, it once again is shown that out of all tested ISPs only T-Mobile Austria is manipulating traffic. Interestingly, this is not the case for all conducted tests but only for a certain subset, amounting to around 11% of conducted tests, not changing throughout the measurement period. Unfortunately, the YAML provided by Tor OONI does not give detailed results on which of the different requests were tampered with.

	Tampering	
	False	True
A1 DSL	1311	
A1 LTE	1414	
Drei	963	
TMA	1050	151
UPC	615	

Table 4.19: Invalid Request line tampering

From inspecting single tests, it shows that:

- Invalid and big request methods were not tampered with.
- Invalid field counts were not tampered with.
- Invalid version numbers were sometimes not transmitted at all.

It is unknown how it is caused, that only some invalid version numbers are not transmitted. Possibly, it could be a programming mistake in the Tor OONI framework. More likely, it could result from some unknown condition at work in T-Mobile Austria's middlebox that blocks some of the random version numbers, but not all.

4.18.3 Blocking: Web Connectivity

The web connectivity measurement did not yield any new results: It only confirmed that all Austrian ISPs block *kinox.to* and *movie4k.to* by using DNS manipulation. Additionally, it reassured, that the websites informing about the censorship for A1 DSL, A1 LTE, T-Mobile Austria and UPC are accessible via their public IP addresses, while the website for Drei is only reachable from within the network.

CHAPTER 5

Discussion

Net Neutrality seems to be respected by Austrian ISPs based on the results. No measurements lead to any detection of traffic shaping of specific protocols.

However, there are a few exceptions to this listed below.

Port 5060 This port is primarily used for the Session Initiation Protocol (SIP) which is essential to VoIP telephony. Port 5060 seems to be blocked for UDP and only for A1 LTE customers. In the measurements, it was not determined if this blockings are due to settings in the default configuration of the modem supplied by the ISP or with the internet connection itself.

Being a mobile provider, A1 would have an incentive to make VoIP calls more difficult to end customers, since they effectively subvert its business plans, moving end customers away from paid telephone calls billed by the minute, allowing to make calls directly over the Internet.

Port 554 This port is essential for multimedia streaming as it allows communication in the Real-Time Streaming Protocol (RTSP). In contrast to port 5060, it was not blocked by any provider but still treated differently for UPC customers. This was measured in that the Time-To-Live values were treated differently than for any other port with UPC customers.

The motivation for this seems unclear since the different treatment did not influence the performance of the port. However, it should be stated that UPC provides a Video-On-Demand streaming service (*ondemand.upc.at*). Since RTSP is used in video streaming, this different treatment could stem from zero-rating or billing purposes.

T-Mobile's Port 80 T-Mobile Austria treats port 80 (HTTP) different from all other tested ports. This shows in different metrics as listed below.

- *TCP4*: Port 80 has different TTL values than any other port with T-Mobile customers. The TTL value of port 80 is generally higher than for other ports.
- *CM7/HTTP7/VS7*: While T-Mobile does not interfere with the transmitted HTTP content, the HTTP headers are changed. HTTP requests gain a *Host* header field if they to not contain one. Additionally, the *Connection: keep-alive* field is added to the HTTP header if it does not yet exist.
- *OONI7*: A fraction of HTTP requests that contain invalid HTTP numbers (e.g. *GET / HTTP/kRm*) are never received by a server when they are sent over T-Mobile's network infrastructure.

  In addition to tampering with the HTTP header fields as mentioned above, T-Mobile streamlines the capitalization of field names, leading to the conclusion, that the whole HTTP header is replaced.
- *Other*: During development or in other tests, it additionally showed that content exceeding the *Content-Length* header field is not transmitted at all. Additionally, any traffic from the receiver of the request to the sender using the same TCP connection is discarded.

Based on these test results, it is safe to assume that these modifications are not done by the modem provided by T-Mobile, but by the ISP itself. This leads to the conclusion, that, at least for port 80, deep packet inspection (DPI) is performed by the provider.

As for the reason, why T-Mobile uses resource-intensive DPI for tampering with HTTP header fields: Adding a *Connection: keep-alive* reduces the time needed for TCP handshakes that would otherwise be used for per-object connections. This strategy is also used by some ISPs in the U.S.[37].

Censorship Austrian ISPs are forced to censor certain domains by court orders. Domains affected by this are e.g. the streaming sites *www.kinox.to* and *www.movie4k.to*. Additionally, all providers with the exception of T-Mobile block *www.thepiratebay.se*.

All ISPs block on a DNS level. Censorship on IP level would be difficult since all of the mentioned sites use the content delivery network *CloudFlare* that is used by many other, legit, websites.

With the exception of UPC, all providers re-route customers trying to access censored websites to custom web pages, informing about the censorship and the reason for the censorship. These custom pages provide transparency to customers. UPC's DNS server returns the non-existing IP *0.0.0.0* for these websites.

Since the information on these custom pages is formulated in a way that shifts the blame away from the ISPs, it seems plausible that ISPs are unhappy with the current way this censorship is done. This is also indicated by the sheer number of court cases in these censorships (e.g. OGH 4 Ob 22/15m, OGH 4 Ob 71/14s, OGH 4 Ob 6/12d among others).

NXDOMAIN rerouting While all other ISPs return the correct *NXDOMAIN*-status code to DNS requests for non-existing domains, Drei answers with an existing IP address from its address space. This address can only be reached from within the Drei network,

As described in the *NDNS7* measurement, this page contains a Google Analytics integration along with a Google custom search form that is pre-filled with the search request. There are no direct advertisements displayed on the page. Following the paper of *Weaver et al.* [35], the reason behind this could possibly be found in monetization. On the one hand, Drei collects data from these pages. On the other hand, the custom search form contains ads provided by Google, possibly directly generating revenue for Drei.

CHAPTER 6

Future Work

Based on the findings in this work, there are many possible alterations and enchantments to measurements possible for future work. Some of the more obvious options are listed below.

Measurment period The measuring period for this thesis was a mere two months, in which some test clients were offline at times or did not conduct usable measurements. In the future, these tests could be run for a longer time, trying to find some long-term patterns.

IPv6 All measurements were done using the IPv4 stack. Some ISPs could alter traffic only for IPv4 or IPv6.

T-Mobile's Port 80 In this thesis, it was concluded that T-Mobile Austria uses some sort of proxy for its port 80. This is based on TTL values and HTTP header tampering.

When inspecting transparent proxies in the U.S., *Xu et al.* [37] found that not only HTTP headers are altered, but also the underlying TCP connections. In the U.S., ISPs for example suppress the *RST*-flag in connections, delay TCP handshakes or split connections.

In the future, it could be tested, if any of these operations are also done by Austrian ISPs, especially T-Mobile.

Greater port range For measuring RTT, TTL and connection speeds, only a handful of hand-selected ports were used due to resource constraints. In future work, other ports could also be included in measurements.

More diversity for headers The *MM7* metric was specially designed to detect traffic shaping in the form of the BingeOn-program of T-Mobile USA. For this, a request for a YouTube-video, that would be treated differently by BingeOn, was used.

However, in the U.S., there are much more hosts[19] that are included in the BingeOn-program.

In future measurements, these hosts could also be tested to detect, if Austrian ISPs e.g. do not slow down YouTube, but slow down other services that were not tested for this thesis.

Full protocol simulations When measuring traffic shaping based on different protocols, these protocols were only simulated by their ports. Any shaping that also inspects the protocol header (e.g. BitTorrent traffic) could therefore not yet be detected.

To remedy this in the future, tests could follow the correct protocol specifications. This could be done by either implementing some minimal stack of the protocol for sending random but syntactically correct data or by replaying traffic that was previously recorded.

Data caps Some ISPs have data plans with a fixed data cap. After customers used up the included volume, the bandwidth is throttled. This should affect all services the same.

However, it is possible that some services are still available with full bandwidth to customers. This was the case with Spotify for Deutsche Telekom[31]. In future tests, this could also be measured.

Variance in hostnames The results showed that Drei does not return a *NXDOMAIN* status for DNS requests of non-existing domains. However, in the test setup, only a static fixed domain name was used. As *Weaver et al.* [35] showed, some ISPs only redirect for certain domain lookups to custom search pages. In future research, the hostnames for non-existing domains should be randomized.

Hostname based caching In contrast to American ISPs[21], Austrian ISPs did not cache static content for the custom sites tested. However, it was not tested, if ISPs cache content of popular websites (e.g. stylesheets for *orf.at*) based on the *Host* HTTP header. This test could also be implemented as a complementary test in future work.

CHAPTER 7

Conclusion

In our research, we showed that net neutrality in Austria is, at least from a technical perspective, mostly respected. The metrics mentioned in chapter 3 were designed to cover a broad spectrum of violations, many of them given in papers of other researchers, mentioned in chapter 2.

The measurement architecture followed a client-server-centric approach, with a central server and five measurement clients, conducting tests in a schedule that was set up using a crontab-style configuration. Each and every test generated a result in the JSON format. This result was then saved on the server-side in a MongoDB database. Besides that, all network communication on both client and server side was recorded for each individual test and saved in a *pcap*-file.

To prohibit interference of foreign clients, the server's firewall was configured to only allow ssh traffic and from certain IPs. Since all clients had dynamic IPs, the current IP was periodically transmitted to the server via ssh, ensuring that only the test clients had access to all server ports at all times.

Using this system, we conducted over 100,000 measurements over the period of two months. We tested for network interference in low-level TCP segments and UDP datagrams, connection speeds and much more. While most of these low-level tests did not yield any significant results, there were the three exceptions mentioned below.

- The difference in TTL for port 80 with T-Mobile clients.
- The difference in TTL fort port 554 with UPC.
- The blocking of port 5060 for A1 LTE clients.

Even though all tests were periodically executed on an hourly basis, 24 times per day, we did not see any differences based on the time of the day or the duration of the measurement period. This held true for round-trip times, connection speeds and other metrics.

When investigating the censored domains *thepiratebay.se*, *movie4k.to* and *kinox.to*, we were able to draw conclusions about the technical background of these blockings. It was detected that all ISPs use DNS for fulfilling their censorship obligations. With the exception of UPC, all ISPs provide information to the user on why a page is censored.

Another result that these DNS tests yielded was that Hutchison Drei does not treat non-existing domains appropriately. Instead of returning the *NXDOMAIN* error code, customers are presented with a custom error-page, containing a Google custom search field and integrated Google Analytics.

For the high-level tests, we also did not find many anomalies. The exception to this was T-Mobile Austria. Our tests showed, that T-Mobile uses some sort of middlebox. This middlebox affects port 80 and changes fields in the HTTP header while leaving the content unchanged. In some cases, this middlebox also caused malformed HTTP traffic to not be transmitted at all.

The motivations for all detected ISP measures were briefly discussed in chapter 5. For T-Mobile's middlebox, it is mentioned, that some U.S. ISPs use comparable transparent proxies to optimize their network infrastructure[37].

Following our research, possible further investigations were introduced in chapter 6. It seems reasonable to at least conduct the existing measurements for a longer time period for detecting any further inconsistencies. Following the results, especially T-Mobile's middlebox, further research in this direction could be conducted, also testing for further anomalies in the TCP traffic, e.g. missing *RST*-flags.

Bibliography

[1] *raw(5) Linux User's Manual.*

[2] Alexa. Top sites in austria, 2016.

[3] Collin Anderson, Philipp Winter, and Roya.

[4] Vaibhav Bajpai, Steffie Jacob Eravuchira, and Jürgen Schönwälder. Lessons Learned From Using the RIPE Atlas Platform for Measurement Research. *SIGCOMM Comput. Commun. Rev.*, 45(3):35–42, Jul 2015.

[5] Hsing Kenneth Cheng, Subhajyoti Bandyopadhyay, and Hong Guo. The debate on net neutrality: A policy perspective. 22:60–82, Mar 2011.

[6] Steven Cherry. The VoIP Backlash. *IEEE Spectrum*, 42(10):61–63, 2005.

[7] Jay Pil Choi, Doh-Shin Jeon, and Byung-Cheol Kim. Net neutrality, business models, and internet interconnection. *American Economic Journal: Microeconomics*, 7(3):104–141, Aug 2015.

[8] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire. Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335 (Best Current Practice), August 2011.

[9] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[10] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, P. Krishna Gummadi, Ratul Mahajan, and Stefan Saroiu. *Glasnost: Enabling End Users to Detect Traffic Differentiation.*, page 405–418. 2010.

[11] Wesley M Eddy. TCP SYN flooding attacks and common mitigations. 2007.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

[13] Arturo Filasto and Jacob Appelbaum. *OONI: Open Observatory of Network Interference.* 2012.

[14] Eletronic Frontier Foundation. Net neutrality, Jul 2016.

[15] Brian Fung. Netflix: We'll pay comcast's ransom, but we shouldn't have to. *Washington Post*, Mar 2014.

[16] Google. Public DNS.

[17] P. Hoffman. SMTP Service Extension for Secure SMTP over Transport Layer Security. RFC 3207 (Proposed Standard), February 2002.

[18] Jacob Hoffman-Andrews. ISPs Removing Their Customers' Email Encryption, Nov 2014.

[19] Arash Molavi Kakhki, Fangfan Li, David Choffnes, Ethan Katz-Bassett, and Alan Mislove. *BingeOn Under the Microscope: Understanding T-Mobiles Zero-Rating Implementation*, page 43–48. Internet-QoE '16. ACM, 2016.

[20] Arash Molavi Kakhki, Abbas Razaghpanah, Anke Li, Hyungjoon Koo, Rajesh Golani, David Choffnes, Phillipa Gill, and Alan Mislove. *Identifying Traffic Differentiation in Mobile Networks.* 2015.

[21] Sheharbano Khattak, David Fifield, Sadia Afroz, Mobin Javed, Srikanth Sundaresan, Vern Paxson, Steven J. Murdoch, and Damon McCoy. Do You See What I See? Differential Treatment of Anonymous Users. In *in Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[22] J. Klensin. Simple Mail Transfer Protocol. RFC 5321 (Draft Standard), October 2008.

[23] Robin S. Lee and Tim Wu. Subsidizing creativity through network design: Zero-pricing and net neutrality. *The Journal of Economic Perspectives*, 23(3):61–76, Aug 2009.

[24] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (INTERNET STANDARD), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936.

[25] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939 (INTERNET STANDARD), May 1996. Updated by RFCs 1957, 2449, 6186.

[26] Gabi Nakibly, Jaime Schcolnik, and Yossi Rubin. Website-targeted false content injection by network operators. *arXiv:1602.07128 [cs]*, Feb 2016. arXiv: 1602.07128.

[27] Body of European Regulators for Electronic Communications (BEREC). All you need to know about net neutrality rules in the eu.

[28] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980.

[29] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[30] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (INTERNET STANDARD), July 2003. Updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164.

[31] Martin Stepanek. Deutsche Telekom drosselt Spotify - Drei berät noch. *Futurezone.at*, Mar 2016.

[32] Markus Sulzbacher. Netzneutralität: Kritik an Spotify-Angebot von "3" wird lauter. *Der Standard*, Jun 2014.

[33] Christoph Sölder, Leonhard Wimmer, Dietmar Zlabinger, Ulrich Latzenhofer, Ursula Prinzl, Philipp Sandner, Lukasz Budryk, and Ulrich Liener. RTR Multithreaded Broadband Test (RMBT) Specification. Oct 2015.

[34] VirusTotal.com. Report for eicar.com, 2016.

[35] Nicholas Weaver, Christian Kreibich, and Vern Paxson. *Redirecting DNS for Ads and Profit.* 2011.

[36] Eddy Willems. The winds of change: Updates to the eicar test file. page 13–15, Jun 2003.

[37] Tobias Flach Ethan Katz-Bassett David Choffnes Ramesh Govindan Xing Xu, Yurong Jiang. Investigating Transparent Web Proxies in Cellular Networks. In *Proceedings of the Passive and Active Measurement Conference (PAM '15)*, March 2015.

[38] Maria Xynou. Tor OONI spec: HTTP invalid request line, May 2016.

[39] Chris Ziegler. T-mobile's "music freedom" is a great feature — and a huge problem. *The Verge*, Jun 2014.

[40] Hubert Zimmermann. OSI reference model-the ISO model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28(4):425–432, 1980.

APPENDIX A

Server configuration

This configuration given in the JSON format is used to configure the test scheduling and the configurations for the single tests. It is stored with the server, the respective parts are transmitted to the client when necessary.

```
{
  "server": {
    "host" : "128.130.204.XXX",
    "port" : 8082
  },
  "mongo": {
    "host" : "localhost",
    "port" : 27017,
    "database": "test-database",
    "test-collection": "tests"
  },
  "enable_client_ip_handling" : true,
  "client_ip_directory": "/home/measurement/client_ips/",
  "tests": {
    "cm7" : {
      "schedule" : "0 * * * * ",
      "duration_seconds" : 170,
      "port" : 80,
      "host" : "128.130.204.34",
      "requests": [
            {
                "resource" : "GET /{test_uuid}/image1.bmp HTTP/1.1"
            },
            {
                "resource": "GET /{test_uuid}/image1.bmp HTTP/1.1"
            },
            {
                "resource": "GET /{test_uuid}/image2.jpg HTTP/1.1",
                "repeat": 3
            },
            {
                "resource" : "GET /{test_uuid}/faultyResponse HTTP/1.1"
            },
            {
                "resource" : "GET /videoplayback?mime=video/webm&dur=610.640&upn=q_PY3To1fWI
                 ↪  HTTP/1.1",
                "header": [
                    "Host: r1---sn-4g5edne7.googlevideo.com",
                    "User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101
                     ↪  Firefox/47.0",
                    "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
```

```
                    "Accept-Language: de,en-US;q=0.7,en;q=0.3",
                    "Accept-Encoding: gzip, deflate, br",
                    "X-Test: {test_uuid}"
                    ]
            }
        ]
    },
    "vs7": {
      "schedule" : "3 * * * * ",
      "duration_seconds" : 45,
      "port" : 80,
      "host" : "128.130.204.34",
      "requests": [
        "GET /{test_uuid}/eicar.exe HTTP/1.1"
      ]
    },
    "http7" : {
      "schedule" : "4 * * * * ",
      "duration_seconds" : 45,
      "port" : 80,
      "host" : "128.130.204.34",
      "requests": [
        "GE/T /{test_uuid}/faultyResponse HTTP/1.1"
      ]
    },
    "tcp4" : {
      "schedule" : "5 * * * * ",
      "duration_seconds" : 110,
      "host" : "128.130.204.34",
      "ports" : [
        80,
        220,
        443,
        554,
        1725,
        1214,
        5060,
        6881,
        8333,
        48123
      ],
      "pings" : 10
    },
    "udp4": {
      "schedule" : "7 * * * * ",
      "duration_seconds" : 110,
      "host": "128.130.204.34",
      "ports": [
        1725,
        5060,
        6881,
        9987,
        48123
      ],
      "packets": 10
    },
    "ndns7": {
      "schedule" : "37 * * * * ",
      "duration_seconds" : 50,
      "requests": [
            {
                "host": "www.orf.at"
            },
            {
                "host": "www.123hjaf9hu32iufhuihoafine.com"
            }
        ]
    },
    "bdns7": {
      "schedule" : "38 * * * * ",
      "duration_seconds" : 50,
      "requests": [
            {
```

```
                "host": "www.thepiratebay.se"
            },
            {
                "host": "www.kinox.to"
            },
            {
                "host": "www.kinox.to",
                "nameservers": ["8.8.8.8"]
            }
        ]
    },
    "syn4": {
      "schedule" : "29 * * * * ",
      "duration_seconds" : 50,
      "host": "128.130.204.34",
      "port": 443,
      "count": 20,
      "source_port": {
        "min": 40234,
        "max": 41453
      }
    },
    "stls7": {
      "schedule" : "30 * * * * ",
      "duration_seconds" : 50,
      "host": "128.130.204.34",
      "ports": [
        8025,
        8026
      ],
      "test_uuid": "{test_uuid}",
      "valid_response": true
    },
    "smtp7": {
      "schedule" : "31 * * * * ",
      "duration_seconds" : 50,
      "host": "128.130.204.34",
      "ports": [
        8025
      ],
      "test_uuid": "{test_uuid}",
      "valid_response": false
    },
    "pop37": {
      "schedule" : "32 * * * * ",
      "duration_seconds" : 50,
      "host": "128.130.204.34",
      "ports": [
        110,
        8110
      ],
      "test_uuid": "{test_uuid}"
    },
    "trac3": {
      "schedule": "39 * * * * ",
      "duration_seconds": 50,
      "requests": [
        {
          "host": "www.orf.at"
        },
        {
          "host": "www.google.at"
        },
        {
          "host": "www.derstandard.at"
        },
        {
          "host": "www.123hjaf9hu32iufhuihoafine.com"
        }
      ]
    },
    "tls4": {
      "host": "128.130.204.34",
```

```
186         "schedule": "36 * * * * ",
187         "duration_seconds": 50,
188         "test_uuid": "{test_uuid}",
189         "host": "128.130.204.34",
190         "port": 443
191       },
192       "tcps4": {
193         "host": "128.130.204.34",
194         "schedule": "9 * * * * ",
195         "duration_seconds": 590,
196         "ports": [
197           {
198             "port": 80
199           },
200           {
201             "port": 443
202           },
203           {
204             "port": 6881
205           },
206           {
207             "port": 48123
208           }
209         ],
210         "http_headers" : [],
211         "packet_size": 4096,
212         "test_duration_ms": 12000,
213         "concurrent": false,
214         "test_uuid": "{test_uuid}"
215       },
216       "mm7": {
217         "host": "128.130.204.34",
218         "schedule": "19 * * * * ",
219         "duration_seconds": 590,
220         "ports": [
221           {
222             "port": 80,
223             "answer_with" : 0,
224             "request_with" : 1
225           },
226           {
227             "port": 48123,
228             "answer_with" : 0,
229             "request_with" : 1
230           }
231         ],
232         "http_headers": [
233               {
234                   "id": 0,
235                   "header": "HTTP/1.1 200 OK\r\nAccept-Ranges:
                      ↪  bytes\r\nAccess-Control-Allow-Credentials: true\r\nAlt-Svc: quic=\":443\";
                      ↪  ma=2592000\r\nAlternate-Protocol: 443:quic\r\nCache-Control: private,
                      ↪  max-age=21293\r\nConnection: keep-alive\r\nContent-Length:
                      ↪  1718030000\r\nContent-Type: video/webm\r\n\r\n"
236               },
237               {
238                   "id": 1,
239                   "header": "GET /videoplayback?mime=video/webm&dur=610.640&upn=q_PY3To1fWI
                      ↪  HTTP/1.1\r\nHost: r1---sn-4g5edne7.googlevideo.com\r\nUser-Agent: Mozilla/5.0
                      ↪  (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0\r\nAccept:
                      ↪  text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\nAccept-Language:
                      ↪  de,en-US;q=0.7,en;q=0.3\r\nAccept-Encoding: gzip, deflate, br\r\n\r\n"
240               }
241           ],
242         "packet_size": 4096,
243         "test_duration_ms": 12000,
244         "concurrent": false,
245         "test_uuid": "{test_uuid}"
246       },
247       "voip7":  {
248           "schedule": "33 * * * * ",
249           "duration_seconds": 170,
250           "host": "128.130.204.34",
```

```
            "ports": [2222],
            "test_uuid": "{test_uuid}",
            "replay_pcap": "./metrics/voip7/rtpstream.pcap",
            "call_duration_ms": 14000
        },
        "ooni7":  {
            "schedule": "40 * * * * ",
            "duration_seconds": 590,
            "test_uuid": "{test_uuid}"
        }
    },
    "dumpcap": {
      "path_to_executable" : "/usr/bin/dumpcap",
      "save_location" : "/home/measurement/pcaps/",
      "interface" : "ens3f0"
    },
    "results": {
      "save_location" : "/home/measurement/results/"
    }
}
```

APPENDIX B

Tor OONI testdecks

The OONI testdecks are used to configure which tests the Tor OONI framework should perform. The *default.deck* file configures the tests, the *citizenlab-urls-global.txt* file contains all domains used for the censorship test.

default.deck

```
1  - options:
2      annotations: null
3      bouncer: null
4      collector: null
5      no-collector: 0
6      no-geoip: 0
7      no-yamloo: 0
8      reportfile: null
9      subargs: &id001 []
10     test_file: manipulation/http_invalid_request_line
11     verbose: 0
12 - options:
13     annotations: null
14     bouncer: null
15     collector: null
16     no-collector: 0
17     no-geoip: 0
18     no-yamloo: 0
19     reportfile: null
20     subargs: *id001
21     test_file: manipulation/http_header_field_manipulation
22     verbose: 0
23 - options:
24     annotations: null
25     bouncer: null
26     collector: null
```

```
27      no-collector: 0
28      no-geoip: 0
29      no-yamloo: 0
30      reportfile: null
31      subargs: [-f, ../deck/citizenlab-urls-global.txt]
32      test_file: blocking/web_connectivity
33      verbose: 0
```

citizenlab-urls-global.txt

```
1 http://kinox.to
2 http://movie4k.to
```