Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

A modular framework for a collaborative solution for privacy-preserving network communication

Christian Brosche

Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

A modular framework for a collaborative solution for privacy-preserving network communication

Ein modulares Framework zur gemeinschaftlichen Lösung privatsphäre-erhaltender Netzwerkkommunikation

Author	Christian Brosche
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Dr. Holger Kinkelin
	Marcel von Maltitz, M.Sc.
Date	August 12, 2015

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, August 12, 2015

Signature

Abstract

In recent years, more and more information has come to light that shows that private citizens are routinely targeted by government surveillance. This surveillance happens in the form of internet traffic inspection. Information that government agencies can gain from this includes website accesses and internet communication – both written and spoken.

There are already multiple approaches that try to help internet users protect themselves from this kind of surveillance. However, even though some of them work well in their restricted use case, there currently is no solution that offers the ability to protect arbitrary traffic. There is also no established solution that provides privacy for real time traffic such as voice over IP.

As a result, there is a need for an approach that helps protect internet users from surveillance, isn't as restricted in regard to protocols and is usable with real time traffic. This paper presents design, implementation and evaluation of the network underlay for a system that provides these features. This system is called *PrivacyBox*. It allows users to transfer both UDP and TCP traffic privately. Other parts of the system are developed in separate theses.

The concept uses a decentralized approach and works for networks of friends/contacts that want to exchange information. It uses layered encryption as well as successive relaying through multiple hops in order to hide information compromising privacy from possible observers. The developed framework intercepts all traffic and thus is able to decide which traffic is supposed to be transferred privately in a transparent way.

Zusammenfassung

Im Laufe der letzten Jahre wurde durch eine Reihe von Veröffentlichungen bekannt, dass auch normale Bürger routinemäßig von Regierungen überwacht werden. Diese Überwachung findet hauptsächlich in Form von Inspektion des Internet-Verkehrs statt. Behörden können daraus unter anderem Informationen zu Webseitenzugriffen und sowohl geschriebener als auch gesprochener Kommunikation erhalten.

Aktuell gibt es bereits einige Ansätze, die den Internetnutzern helfen wollen, sich vor solcher Überwachung zu schützen. Obwohl diese Ansätze zum Teil guten Schutz für den von ihnen gewählten Anwendungsfall liefern gibt es zur Zeit noch keine Lösung, die beliebigen Internetverkehr schützen kann. Darüber hinaus gibt es auch noch keine etablierte Lösung, die Echtzeittraffic wie z.B. Voice-over-IP schützen kann.

Deshalb ist ein Ansatz nötig, der Internetnutzern dabei hilft, sich vor Überwachung zu schützen, der weniger Einschränkungen in Bezug auf kompatible Protokolle mit sich bringt und der auch Echtzeittraffic schützen kann. Die vorliegende Arbeit stellt Design, Implementierung und Evaluation der Netzwerk-Komponente eines Systems vor, welches diese Eigenschaften besitzt. Dieses System heißt *PrivacyBox*. Es ermöglicht die Übertragung von TCP und UDP traffic auf eine Art und Weise, die Privatheit garantiert. Weitere Komponenten dieses Systems werden in seperaten Arbeiten entwickelt.

Das hier vorgestellte Konzept verwendet einen dezentralen Ansatz und bietet Netzwerken von Freunden bzw. Kontakten die Möglichkeit, auf private Art und Weise Daten auszutauschen. Es verwendet mehrere Verschlüsselungsschichten und sukzessives Umleiten über mehrere Stationen, um private Informationen vor möglichen Beobachtern zu verstecken. Das entwickelte Framework inspiziert jeglichen Internetverkehr. Deswegen kann es ohne nötige Interaktion mit dem Nutzer entscheiden, welcher Traffic normal und welcher traffic mit Privatheits-Eigenschaften übertragen werden soll.

Contents

1 Introduction 1
- 1.1 Problem statement . . . 2
- 1.2 Goals of the thesis . . . 3
- 1.3 Research questions . . . 3
- 1.4 Outline . . . 4

2 Background 5
- 2.1 Routing . . . 5
- 2.2 Network Security . . . 6
- 2.3 Privacy . . . 7
- 2.4 Basic technologies . . . 9
  - 2.4.1 Public Key Infrastructure . . . 9
  - 2.4.2 OpenVPN . . . 9
  - 2.4.3 TLS/DTLS . . . 10
  - 2.4.4 libnetfilter_queue . . . 11
  - 2.4.5 Twisted . . . 11

3 Related work 13
- 3.1 Bitmessage . . . 14
- 3.2 Anonymous Proxies . . . 15
- 3.3 Tor . . . 16
- 3.4 VPNs . . . 18
- 3.5 Summary . . . 18

4 Analysis 19
- 4.1 Scenario . . . 19
- 4.2 Attacker model . . . 20
  - 4.2.1 Passive attacks . . . 20
  - 4.2.2 Active attacks . . . 21
- 4.3 Requirements . . . 21
  - 4.3.1 Functional requirements . . . 21
    - 4.3.1.1 Traffic interception . . . 21

4.3.1.2 Traffic differentiation . . . 21
4.3.1.3 Application layer protocol extension . . . 22
4.3.1.4 Providing Privacy . . . 22
4.3.1.5 Addressability . . . 22
4.3.2 Non-functional requirements . . . 22
4.3.2.1 Performance . . . 23
4.3.2.2 Modularity . . . 23
4.3.2.3 Extensibility . . . 23
4.4 Solution processes . . . 23
4.4.1 Intercepting traffic . . . 24
4.4.2 Differentiating traffic . . . 24
4.4.3 Providing privacy . . . 25
4.4.4 Addressing users . . . 26

5 Design 27
5.1 Overview . . . 27
5.2 System design . . . 29
5.2.1 Traffic interception module . . . 30
5.2.2 Traffic inspection module . . . 31
5.2.3 Application layer protocol modules . . . 31
5.2.4 Tunnel establishment module . . . 31
5.2.5 Peer information module . . . 33
5.2.6 Server module . . . 33
5.3 Security . . . 34

6 Implementation 37
6.1 Traffic interception . . . 37
6.2 Traffic inspection . . . 38
6.3 Application layer protocol modules . . . 38
6.3.1 Abstract overview . . . 38
6.3.2 Exemplary module: HTTP . . . 39
6.4 Traffic relaying . . . 40
6.4.1 Management module . . . 40
6.4.2 Tunnel module functionality . . . 40
6.5 Server module . . . 41

7 Evaluation 43
7.1 Performance . . . 43
7.1.1 Latency . . . 43
7.1.1.1 Tunnel length . . . 43
7.1.1.2 Simultaneous connections . . . 48
7.1.2 Packet sizes . . . 48

7.2 Security and privacy 50
7.2.1 Passive attacks 50
7.2.1.1 Passive local attacker 50
7.2.1.2 Passive global attacker 51
7.2.2 Active attacks 52
7.3 Summary 54

8 Conclusion and Outlook 55

Bibliography 57

List of Figures

1.1 NSA slide mentioning PRISM and Upstream 2

2.1 IPv4 header 6

4.1 Abstract overview for the scenario 20
4.2 Bamboo-like tunnel 25
4.3 Telescope-like tunnel 26

5.1 Scenario with module interaction 27
5.2 Overview over an exemplary setup 28
5.3 PrivacyBox modules 30
5.4 Tunnel management module 32
5.5 Client tunnel establishment 35

7.1 Effects of delay 44
7.2 Intracontinental one-way delay for TCP traffic 45
7.3 Intercontinental one-way delay for TCP traffic 46
7.4 Intercontinental one-way delay for UDP traffic 46
7.5 Intracontinental one-way delay for UDP traffic 47
7.6 One-way delay for simultaneous connections 48
7.7 Packet sizes 49
7.8 Overhead 50

List of Tables

3.1 Overview for presented services 18

7.1 Overview for presented services – revisited 54

Chapter 1

Introduction

Ever since the public disclosure of secret NSA documents by Edward Snowden there has been an increased awareness of privacy in the Internet. Many journalists and media organizations started covering the leaks and tried to discern the extent to which the US agency was spying on ordinary people.

One system that received a lot of attention is *PRISM*. It allows the US agencies to receive information from private companies such as Google, Yahoo and Facebook. This is something that internet users can easily understand and be angry about: By using Google's mail service you shouldn't automatically provide all your emails to the NSA, too.

What is at least equally alarming is another program that was mentioned on these slides, but wasn't covered as extensively: It is called *Upstream* and it is tasked with the "collection of communcations on fiber cables and infrastructure as data flows past." (cf. Figure 1.1) [1].

This program is proof of the NSA's capability of intercepting all traffic going into and out of the country, inspecting and possibly storing it [2]. The intercepting capabilities of the NSA are not limited to domestic traffic though, as they collaborate with foreign government agencies in order to get access to certain traffic intercepted by these agencies, too. One example of such a foreign agency is the german BND. The extent to which the BND provided the NSA with access to traffic is currently investigated by an NSA investigation committee (*NSA Untersuchungsausschuss*) [3] appointed by the German Bundestag.

When faced with the fact that the internet traffic of ordinary internet users is routinely intercepted, analyzed and possibly stored the question arises of how users can try to retain their privacy while using the internet.

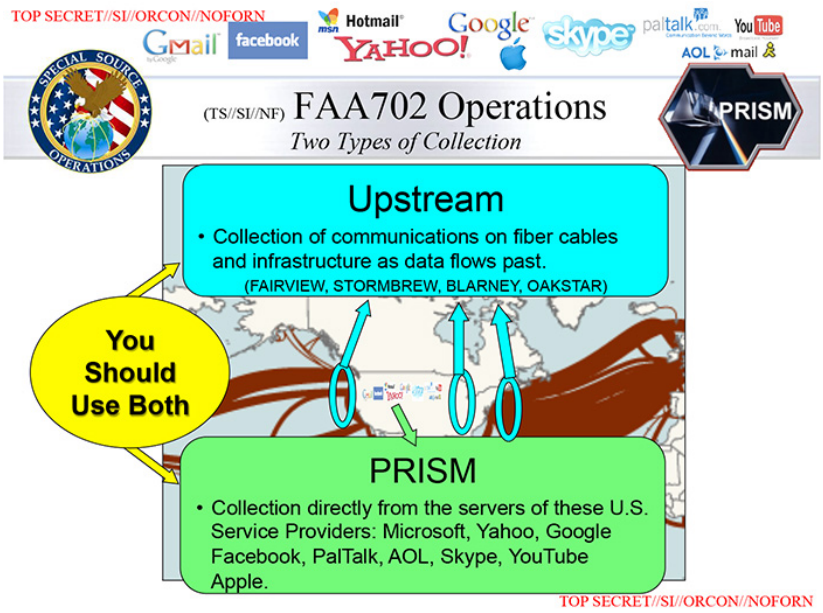

Figure 1.1: NSA slide mentioning PRISM and Upstream

1.1 Problem statement

There are already a few (partial) solutions for this problem. Using encryption when transferring data over the internet is a basic one. HTTPS is one example for this. Usually, the decision if encryption is used or not does not lie in the hands of the users though. An internet service has to provide this option. Additionally, this only protects the content that is being transferred through the internet. Metadata such as IP address and time is still available in the clear. The NSA still knows who accesses which content, when and from where.

What is needed is a way to both hide the content and the metadata. There are already various approaches that try to do this. But none of the currently available ones provide a comprehensive solution enabling a user to gain privacy for all of his or her internet traffic. Tor [4], the most prominent current approach only solves this problem for TCP-based protocols such as HTTP.

One major use case that is lacking in a privacy providing solution is voice over IP (VoIP) communication. Even though more and more telecommunications companies strive to switch over to VoIP completely (cf. [5]), as of yet there is no reliable way to establish a VoIP call during which both content and metadata stay hidden.

1.2 Goals of the thesis

This thesis will present the core for a Tor-like network overlay through which TCP- and UDP-based application layer protocols can be tunnelled. It focuses on tunnel establishment and is called *PrivacyBox*. An HTTP-module that allows accessing a webserver through the network overlay will be developed as a proof of concept. Additional modules for application layer protocols [6] and a solution for address resolution as well as authenticating the overlay network participants [7] are developed in seperate theses.

1.3 Research questions

The most basic question one has to ask is: *How can a privacy providing peer-to-peer network underlay be built*? From this, several follow-up questions arise:

- **To what extent do existing technologies solve the problem?**
  Some existing technologies have goals similar to the ones set for the system introduced in this thesis. These technologies will be examined for their advantages and disadvantages.
- **What exactly does the network need to provide?**
  In order to reach the goals, it has to be clear what attackers the system faces and what functionality it needs to provide.
- **What can the architecture look like?**
  The architecture of the system should be designed in a way that best supports its goals. It will be described accordingly.
- **How can the traffic be intercepted?**
  The system needs to be able to intercept all traffic coming from a device that wants to use the privacy network.
- **How can the system differentiate between normal and privacy-traffic?**
  As the system intercepts both traffic that is bound for the privacy network and normal internet traffic, there needs to be a way that allows distinguishing the two.
- **How can the tunnels be established?**
  Tunnels are the privacy providing connections between multiple PrivacyBoxes. How they can be built in order to provide the properties essential for this system is a central question.
- **How does address resolution work? (cf. [7])**
  Users of the system need to be able to connect to other users. For this it is necessary to resolve the addresses of other participants.

- **How can participants be identified and authenticated? (cf. [7])**
  Users that want to exchange information need to be able to identify and authenticate each other in order to be cartain that the exchange is happening with the right contact.

1.4 Outline

Chapter 2 will provide some basic background information that is necessary in order to understand the rest of the thesis. Chapter 3 discusses related work and introduces concepts such as VPN (section 3.4) or Proxies (section 3.2), but also tools like Bitmessage (section 3.1) and Tor (section 3.3). Chapter 4 presents a thorough investigation of what is needed in order for the presented system to work. Chapter 5 introduces the system design based on the information of the previous chapter. Chapter 6 explains the concrete implementation of the system following this design. In order to be able to make a statement about the extent to which the introduced system succeeds, Chapter 7 presents the results of an evaluation of the system both in regard to performance (section 7.1) and privacy/security (section 7.2). The final chapter 8 concludes the thesis, summarizes its insights and provides a perspective for further work.

Chapter 2

Background

This chapter will provide an introduction to some basic topics related to this thesis. This will help understanding the later chapters.

The chapter starts with explaining the way data is transferred in the internet and also highlights certain special cases in this area. After that, there will be a section that gives an overview of some ways in which the data that flows through the internet can be provided with certain security properties. The last part of this chapter then provides a description of some ways that can reduce an internet user's online privacy.

2.1 Routing

Letters and parcels which are sent through the mail service need unique identifiers for their sender and recipient in order to be delivered to the correct address. In the same way, source and destination of packets sent through the internet also need to be uniquely identifiable. The two main protocols that are currently responsible for this task are the Internet Protocol [8] (IPv4) and the Internet Protocol, Version 6 [9] (IPv6). Both protocols use globally unique identifiers in order to ensure that packets are sent to the right destination. Figure 2.1 shows the content of an IPv4 header. It contains more information than just the IP addresses, but for the purposes of this thesis, it is enough to focus on them.

An IP address is a number that is assigned to a device. Most IP addresses are assigned by internet service providers. IP addresses can be dynamic, but in order to be globally routable, they have to be globally unique at all times. Given an IP address, a device that wants to send data to another device can assemble the IP header and send the packet on its way. During its travel, the packet is relayed by routers until it reaches its destination. Every router inspects the destination IP address in the IP header and decides on the

next hop based on it. In case of transmission problems, routers inform the sender of the packet by sending a message back to the source address listed in the IP header.

This setup implies that every router between sender and recipient knows who - or more specifically which IP - is exchanging data with whom. Beyond that, routers are not restricted to only accessing the data in the IP header. For various reasons it might make sense for an ISP or other actors to try and get more information out of the traffic by inspecting other headers or even the payload. This is called *Deep Packet Inspection*.

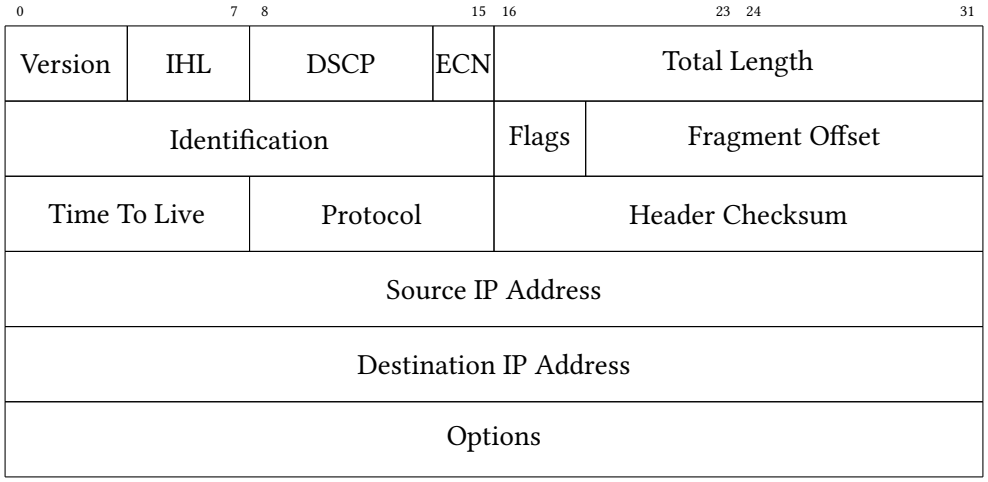

Figure 2.1: IPv4 header

2.2 Network Security

The modern internet is a dangerous place. News about corporations being hacked are commonplace and information can leak in an abundance of ways. Some threats that have to be faced are (cf. [10]):

- Spoofing: An entity claims to be another entity.
- Eavesdropping: An entity accesses information not intended for it to read.
- Loss or modification of information: Data is altered or destroyed.
- Forgery of information: An entity creates information in the name of another entity.
- Denial of service: An entity tries to reduce the availability of a system.

The approaches brought together under the term *Network Security* try to counteract these threats. In order to do this, they try to achieve several security goals (cf. [10]:

- Confidentiality: Only the intended recipient should be able to read transmitted data.
- Integrity: It should be possible to detect modification of data.
- Authenticity: It should be possible to verify the origin of transmitted data.
- Availability: Services should be accessible at all times.

Confidentiality, Integrity and Authenticity can be achieved by employing cryptographic techniques, while guaranteeing availability is almost impossible. If the data sent from one entity to another through the internet is not supposed to be readable for anyone in between, it can be encrypted in transit. For Integrity, cryptographic hash functions can be used. Integrity and Authenticity are mostly provided at the same time, either by using Message Authentication Codes or by using digital signatures over hashes. Availability depends more on efficient implementations and measures that try to recognize attacks like denial of service.

2.3 Privacy

Privacy is something many people inherently value. Be it an entry in a diary, a letter to a far away lover or the balance of a bank account - there are a lot of things where privacy can matter.

According to a survey done by Statista in Germany in 2011 [11], 61% of internet users take special care about internet privacy while 33% at least view it as important. Two studies done by Pew Research in 2015 come to similar conclusions and find that “Americans feel privacy is important in their daily lives” [12].

Nowadays, a big amount of once assumed to be private interaction happens on the internet. We chat, post and talk with each other through a medium that makes it possible to record each and every of these interactions.

Comparing the amount of information that can easily be gathered about one single individual in pre-internet and current times shifts the weight heavily towards the present. Internet traffic provides an abundance of information, can be easily recorded and searched and theoretically be stored for an unlimited amount of time. According to leaked NSA slides [13], a system called *XKEYSCORE* stores all collected content for 3-5 days and all collected metadata for up to 30 days. This “gives analysts unique access to terabytes of content and meta-data” [13]. The amount of time all content and metadata can be stored increases with advances in data storage technology. An internet user has almost no way of knowing if his or her traffic is being recorded and is very limited in trying to avoid it.

There are two kinds of data that can leak private information and they can be compromised in two different ways. The two kinds of data are packet headers - the metadata - on the one hand and the information that is contained in the application layer data on the other.

Application layer data can provide infinite possibilities for compromising privacy. As anything can be the content of an application layer packet, any kind of information could be contained in one. An example for this is HTTP traffic. By using cookies and other technologies, a user can be tracked during visits to different websites without noticing. This allows online advertising companies to target the ads the user sees in a more precise way and thus be able charge more money for ad impressions. How much private data application layer data leaks is completely dependent on the protocol and its use though. No website is forced to employ these techniques and gather this data. Application layer data does not inherently contain private information. Its content also varies very strongly, which makes it a lot harder to automatically make meaning of it.

In contrast to that, the *data contained in headers* is needed for the internet to function. In many cases it can also be used to compromise privacy. A prominent example for that is the IP address that is contained in an IP header. As already mentioned, it needs to be globally unique at the time it is used. This leads to the ability to unambiguously address computers on the internet which is very desirable. A side effect of this is that everyone that monitors the traffic knows exactly when which IP address exchanged information with which other IP address. If the entity, monitoring the traffic, can also connect the IP addresses to persons or websites, for example, this already reveals a lot of information.

The two ways in which both application layer data and metadata can be compromised are *during transit* and *at the end hosts*. Generally both kinds of data can be accessed in both ways.

Preventing these attempts naturally requires different courses of action. First of all, application layer data can be encrypted in transit. Assuming that current encryption technology can not be broken by even the most powerful attacker, this makes it unreadable for anyone but the sender and the recipient. In order to additionally keep private data from leaking to the recipient, a user can first of all be careful not to put it in there. He or she can also employ various techniques to try and remove data that could possibly uniquely identify him or her. One example for this in a web context would be deactivating flash and/or javascript, another example would be the use of certain browser plugins.

Preventing metadata from being accessible and sent through the internet is not possible. If the source IP address of a packet is not the one that is currently owned by the machine sending the packet, no answer will ever arrive. In this case the approach is to try and make the information that tells an observer that one entity exchanged data with another entity less meaningful. One example for this is to undergo steps that prevent the observer

from being sure if the observed recipient was really the recipient of the content of the message and the observed sender was really the inital sender of the message.

2.4 Basic technologies

The following chapter presents various approaches that try to improve internet user's privacy. Before that, this section introduces concepts and terminology developed in previous work that this thesis builds upon.

2.4.1 Public Key Infrastructure

The system developed in this thesis builds upon X.509 [14] certificates for encryption and authentication. However, certificates aren't issued centrally in the same way as for example the certificates issued by certificate authorities that are mostly used for HTTPS-servers. Instead, every user operates his or her own private certificate authority using tools like OpenSSL [15].

This certificate authority is used in order to issue certificates for all devices in the local network that need to be authenticated by other users. This may just be client devices like PCs or smartphones, but could also be something like a local webserver. By issuing a certificate for these devices, they can be authenticated as being run by the same user that operates the issuing certificate authority.

This alone does not help anyone authenticating any device yet. For this to work, a trust exchange is needed. This means that two friends or acquaintances can conduct a procedure during which they authenticate each other and exchange certificates for their certificate authorities (cf. [16]).

These certificates then act as something similar to root certificates. If a service or client run by one contact wants to authentify itself to the other contact, the second one can just check the provided certificate against the certificates of all his contacts. If the provided certificate is valid and was issued by a certificate authority run by one of the contact's, the authentication can be accomplished.

Of course, this system entails the fact that only contacts that know each other can authentify each other. In a system that is supposed to provide privacy to its users, this is a desirable property.

2.4.2 OpenVPN

OpenVPN is a software that allows the creation of virtual private networks. It is open source and can be used for both connecting single clients and whole networks [17].

OpenVPN makes it possible to exchange data with a remote device or network as if the remote device or network were directly connected to the local device using OpenVPN.

It creates a virtual TUN interface in case of an IP tunnel (a routed tunnel) or a TAP interface in case of an ethernet tunnel (a bridged tunnel) [18]. These interfaces look like regular network interfaces to the operating system, but push the received packets into userspace. The OpenVPN software running in userspace then opens these devices like a file and thus is able to read from and write to them. Additionally, it creates a UDP (or possibly TCP) connection to a remote OpenVPN device. Packets entering the TUN (or TAP) device will then be sent over the UDP (or TCP) connection, received at the other end and finally written to the TUN (or TAP) interface there. This creates a setting in which both machines seem to be directly connected or connected to the same local network, even though the traffic is sent through the open internet.

OpenVPN doesn't only provide the ability to create these seemingly local network connections, but it also equips the traffic that is sent through the internet with encryption and authentication. This works by either using pre-shared static keys or TLS. For TLS, OpenVPN relies on the OpenSSL library. Even though OpenVPN is mostly used over UDP it doesn't use DTLS but creates a reliable UDP connection for the TLS connection.

2.4.3 TLS/DTLS

TLS [19] and DTLS [20] are cryptographic protocols that provide encryption, authentication and integrity to TCP and UDP connections respectively. TLS is the successor of SSL, which was first introduced by Netscape in 1995 [21]. The current version of TLS is 1.2. DTLS was developed in 2004 [22] in order to provide UDP with the security features that TLS offers. Its current version is DTLS 1.2.

TLS and DTLS provide their features by building on various cryptographic concepts. These include X.509 certificates, symmetric and asymmetric cryptography and message authentication codes.

The DTLS handshake and protocol are conceptually very similar to TLS, but employ various additional techniques in order to overcome problems resulting from the unreliable nature of UDP.

A wide variety of cryptographic algorithms is supported. Which algorithms are used depends on which are supported by the devices doing the handshake and are agreed upon during it. The handshake might vary in certain aspects depending on the algorithms used and the extent of authentication required (none, one-sided or two-sided). This is reflected by the chosen ciphersuite. While developing and testing the prototype, the ciphersuite, *TLS_RSA_WITH_AES_256_CBC_SHA256* was used.

The name of the ciphersuite includes the various algorithms used. RSA is used for

asymmetric encryption, AES-256 in cipher block chaining mode is used for symmetric encryption and SHA-256 is used for the MAC, which provides integrity and authentication.

X.509 certificates include various information about the party that they are issued for. Most importantly they include a public key of their owner as well as a signature by a trusted third party, in this case called a certificate authority (CA).

When starting a session that is secured by TLS or DTLS, a handshake takes place. During this handshake, various parameters as well as the certificates get exchanged. Then, both parties create various cryptographic keys. These keys are then used to encrypt the following data packets and to verify their integrity and authenticity.

For the implementation of secure connections between PrivacyBoxes, the default python ssl module is used in order to bring TLS to TCP connections. As there is no standard python module that supports DTLS, the PyDTLS package [23] is used for securing UDP connections.

2.4.4 libnetfilter_queue

libnetfilter_queue is a library providing user space access to network packets queried by the linux kernel [24]. It allows inspecting and reacting to the queried packets with a userspace program.

This works by creating iptables rules that queue the desired packets. The userspace program can then receive all packets forwarded by the kernel to the queue by opening a socket on it.

The userspace program can decide what to do with the packet. Options include dropping or accepting the possibly modified packet.

In order to use libnetfilter_queue with python, the *nfqueue-bindings* [25] package is used. It provides bindings to the libnetfilter_queue library for python and perl. For processing a packet, a callback function is passed to the queue. It is called for every packet and receives the content of the packet as a parameter.

2.4.5 Twisted

Twisted [26] is an event-driven networking framework for python. It provides high-level abstractions for easy and fast development of efficient networking applications, but also allows lower level interaction with network sockets.

The central element of the twisted framework is the reactor. This reactor follows

the reactor design pattern [27] which allows it to handle concurrent connections synchronously.

In addition to the reactor, twisted uses two other central concepts: factories and protocols. Factories are a design pattern used in object oriented programming. A factory is itself an object, which is used to create other objects.

The twisted way of writing a networking application is the following: A *protocol* describes how a server or client is supposed to react to incoming packets or established connections. The protocol might inspect the content of the packet and react depending on it, or it might always react in a predefined way independent of the content. A *factory* [28] has a reference to one specific protocol. If necessary, the factory creates an object of that protocol, for example when a new incoming connection is established on a server. But before this can happen, the factory needs to be subscribed to the *reactor*. In order to do this, an instance of the factory is created and subscribed to the reactor through one of various methods, depending on the type of connection.

If there are multiple connections simultaneously, the factory just creates new protocol objects for every connection and the reactor handles the correct processing of the packets.

Because of its event-driven nature, twisted relies heavily on callbacks. In fact, for an application to properly work with twisted, the whole application needs to be developed in an event-driven way. The reason for this is that the inital call that starts the reactor is of a blocking nature. This means, that after the reactor has been started, only code that somehow subscribes to it and is called from it via a callback, is executed.

In the case of the PrivacyBox, this included the fact that the netfilter_queue had to be subscribed to the reactor via a socket.

Chapter 3

Related work

There are already various approaches that try to preserve privacy on the metadata-level, even though the architecture of the internet makes this a difficult task. Their common goal is hiding or obscuring the fact that an entity is accessing a certain service or connecting to a certain other entity. In some cases, they also provide additional encryption whereas in others, the data is left unchanged.

An attacker, who is interested in the data found in packet headers, wants to know who accessed which service at what time. In general, there are two ways to try and prevent the possibility of making reliable conclusions regarding this information based on the packet metadata:

- Proxying: The user sends encrypted data to a third party which relays it to the target service and back. Depending on the kind of third party, this can make it hard for the observer to deduce which user connects to which service.
- Multicasting/Broadcasting: All data is encrypted and sent to a group of/all users within a privacy providing system.

With proxying, the observer can know that a user sent something, but not as easily know to whom; the observer can also know that a service received something, but not as easily know from whom. Both of these conclusions can be made harder or even prohibited by using additional techniques.

With Multicasting/Broadcasting, an observer can know that a user sent something but not who received the content. Or to be more precise: the observer sees that multiple/all other users received something, but he or she can't be sure who was the intended recipient. This is generally combined with cryptographic techniques, which also makes sure that only the intended recipient is able to make sense of the received data.

In the following, various existing approaches to provide privacy for (parts of) a user's internet traffic will be presented. An approach relying on Multicasting/Broadcasting

will make a start, followed by approaches relying on proxying.

3.1 Bitmessage

Bitmessage [29] is a messaging system that provides privacy to all messages exchanged with it. It does so by using the Broadcast technique.

The goal of Bitmessage is to provide "a communications protocol and accompanying software that encrypts messages, masks the sender and receiver of messages from others, and guarantees that the sender of a message cannot be spoofed, without relying on trust and without burdening the user with the details of key management." [29]. The paper states email, PGP/GPG, the X.509 system and Tor as technologies that it wants to improve upon.

The stated problems with these technologies are that they are either insecure (email), difficult (PGP/GPG) or vulnerable (Tor, because it uses X.509). In general the argument is that one cannot anonymously send secure mail-like messages by using these existing and widely used technologies. Even in a setting of using PGP/GPG for encryption and Tor for anonymity, the connection to the mailserver is secured through HTTPS/TLS, which relies on the X.509 system. The X.509 system is considered not trustworthy by the paper's author. As reason he states the big number of root certificates that are inherently trusted by browsers and operating systems. Specifically the fact, that the compromise of one of the Certificate Authorities owning such a root certificate would possibly compromise the whole system.

Bitmessage provides an approach that tries to solve these problems. Bitmessage's focus is on providing anonymity to its users.

Bitmessage uses public key cryptography for encryption. A user can create one (or multiple) identities. Each identity will have its own public-private-key pair. The identity will be represented in the network by a hash from the public key. This hash also acts as the Bitmessage-address. As a result, identities cannot be spoofed. They also remain anonymous as long as no connection between hash and real identity is made. Authentication is possible by comparing hashes, in a manner similar to PGP/GPG.

The Bitmessage client is connected to between eight and 50 other clients at any time. As every client in the Bitmessage network gets every message, the connected clients send each other all messages that they receive and that the other party doesn't already have. This way the network is flooded with the messages until every peer is in posession of it. There are four kinds of objects that can be sent through the Bitmessage network: public key requests, public keys, person-to-person messages and broadcast messages.

Knowing the ID of a user, his or her public key can be queried. A message is sent by encrypting it with this public key and sending it into the network. Every peer, including

the intended recipient, receives the message. Every peer will also try to decrypt the message, but only the intended recipient will be successful.

As a result of every Bitmessage user receiving every message that is sent, the scalability of the system has to be considered. In order to cope with these issues, Bitmessage uses streams: Once a certain threshold is reached, the network is split up into multiple streams. Messages are then only sent to every peer inside the stream of the recipient. This sets an upper bound for the amount of possible recipients of a message and thus may hinder privacy. The scalability issues would also be amplified considerably by VoIP traffic, as every user in a stream would need to receive and try to decrypt all telephony traffic sent in the stream.

Bitmessage is still in its beta phase and thus not ready for use in a scenario where privacy or anonymity are of paramount importance. Examples for weaknesses are:

- The use of a traditional public-key encryption scheme without perfect forward secrecy. Particularly important in this case, as everyone receives every message without having to be able to intercept it in the first place. If a private key is compromised, an attacker can decrypt all previously received messages addressed to this private key.
- No link layer encryption. This leads to observers being able to see that a participant sent a certain kind of object. If no such object was previously seen sent to the participant for relaying, the observer can be certain that the participant is the originator. Depending on the type of object this may lead to a privacy compromise.

Bitmessage shows an interesting approach to providing privacy in the internet. Given the existing weaknesses and a considerable slowdown in development, it may have to be considered more as a proof of concept than a real usable approach.

In the context of this paper the Bitmessage approach does not provide the properties needed to fulfill the goals. The main reason is that Bitmessage is not suited for real-time traffic. The amount of time it takes for a message to reach its destination can vary greatly. Additionally a proof of work has to be calculated when sending a message which further lengthens the process. Bitmessage also does not support privacy for arbitrary kinds of traffic.

3.2 Anonymous Proxies

One tool to achieve a basic level of privacy are *Anonymous Proxies*. In general, they work in the following way: A user connects to the proxy, sends the data to the proxy and the proxy then relays it to the intended recipient. A response is sent to the proxy, which in turn relays it to the origin of the first connection. This prevents the operator

of the target service from figuring out who is accessing the service. If the connection from the initiator to the anonymous proxy is encrypted, it also prevents an observer from easily figuring out which service the initiator is accessing. The reason for this is that it is assumed that the anonymous proxy is relaying a lot of traffic for a lot of different initiators and that this makes it hard to connect the encrypted traffic from the initiator to the relayed traffic from anonymous proxy to destination.

The big problem with anonymous proxies is that the user has to put a lot of trust into the provider of the proxy. At the proxy, all traffic is visible. If the proxy is somehow compromised, this puts the user in an even worse spot than without an anonymous proxy, because potentially all traffic is now sent through the proxy and conveniently accessible for the attacker. Beyond that, an attacker might be able to correlate connections from a user to a proxy with connections from the proxy to the endhost even if the proxy is simultaneously used by many differend users. Other problems are that these proxies mostly only work for selected traffic and that each application has to be configured and configurable to work with them.

3.3 Tor

A concept that builds upon the idea of anonymous proxies and is currently the most popular of the privacy tools available [30] is *Tor* [4]. Fundamentally, it uses multiple anonymous proxies connected in series. It uses this setup combined with a number of other techniques in order to achieve various privacy features. In the following, the central techniques and their purpose will be explained.

Tor introduces an overlay network that allows clients that connect to it both to access websites and TCP-based services (as well as DNS) in the traditional internet as well as those that are hosted inside the Tor overlay network. When accessing a website or a service in the traditional internet through Tor, a user can be fairly certain that the fact of this access is not easily visible for anyone observing his or her traffic. A service that is hosted inside the Tor network also achieves an amount of privacy for itself, as it is made hard to find out the original IP address of the service. But how does it work exactly?

Tor builds upon multiple building blocks that interact in a way that achieves certain privacy features. These building blocks are:

- Onion Proxies: Run on the user's device.
- Onion Routers: Are responsible for relaying data.
- Hidden Services: Provide services inside the Tor network.

All Onion Routers (OR) in the Tor network are connected to each other. Some of the

ORs also act as *directory servers* from which information about available ORs can be requested. An Onion Proxy (OP) can connect to an OR and request that a circuit is started. This means that the OP and the OR start a cryptographic exchange that results in both sides being in posession of a key that is used to encrypt all further traffic. Once this is done, the OP can request the circuit to be expanded by telling the OR to start a new exchange with a second OR. In order to do this Tor introduces its own packets, called *cells*. This enables ORs to distinguish the purposes of received packets. If the OR wants to extend the circuit, a corresponding cell is sent. After the second exchange - in which the first OR acts purely as a relay for encrypted traffic - is done, the OP has agreed on two different keys, one for each OR. The circuit can be extended further analogously.

Tor uses a minimum of three relay nodes. Once a long enough circuit has been established, data can be sent through it. The data is thus encrypted multiple times, once for every relay node. There are also multiple cells added, because every relay node needs to know what to do with the received packet. If a user accesses the traditional internet via Tor through three relay nodes, the first relay node receives some data, decrypts it, sees that the packet is supposed to be relayed and relays it to the next hop. This is done at every following relay node. In the end, the *real* traffic is sent from the last relay node to the intended recipient - for example a web server. Only the last relay node can see the application layer data.

This way, the operator of the website or internet service only sees the connection originating from the last relay node and does not know anything about the real sender. An observer also has a hard time following the traffic flow, because a relay node has connections to multiple other relay nodes. If the observer sees traffic entering the Tor network from a target, he or she generally can not be sure to which second and third hop this traffic is relayed and thus not follow the traffic to its destination.

When accessing a hidden service, an additional concept comes into play: *rendezvous points*. Rendezvous points are ORs that are advertised by the hidden service through Tor itself. The hidden service has standing circuits to all its current rendezvous points. An OP that wants to access a Hidden Service can find out which rendezvous points are available and build a circuit to one of them. This way neither the Hidden Service nor the OP know with whom they are exchanging data.

Tor improves upon many of the problems anonymous proxies bring with them. The service is distributed and there is no single entity having access to all relayed traffic. Correlation is also made considerably harder because the traffic would need to be correlated at multiple hops. Tor still only works for selected traffic though and needs applications to support certain protocols in order to use it.

Service	Type of traffic	Real-time	Third party has access
Bitmessage	Only Bitmessage	No	No
Proxies	Limited	Yes	Yes
VPN	All	Yes	Yes
Tor	Only TCP	No	No

Table 3.1: Overview for presented services

3.4 VPNs

VPN providers like *Hide My Ass* [31] and similar services try to provide a solution that is conceptually similar to anonymous proxies. The user still connects to a server which relays all data that it receives to the originally intended destination. Here, encrypting traffic to the VPN servers is the default. Most importantly though, VPN software can create a virtual network device which allows sending all traffic leaving a user's device through the proxy without the need to configure anything in client applications.

The traffic is still visible at the server on which the VPN service is running. In this regard, VPNs have the same problem as anonymous proxies: A compromise of the VPN server leads to easily accessible user traffic. As with an anonymous proxy, using a VPN also still might allow an attacker to correlate connections.

3.5 Summary

The presented solutions all have their use cases in connection to that their advantages and disadvantages when it comes to certain requirements.

Bitmessage provides privacy through broadcasting or multicasting. But the provided privacy is limited by the stream sizes. Beyond that, it has issues regarding its use of encryption. It is also not usable for real time communication.

Proxies and VPN services take a simple relaying approach. This might achieve some basic privacy when faced with weak attackers, but it can not be considered a real solution: Both approaches entail the fact that a third party has access to all traffic sent and received. This entails the necessity of trusting the third party in multiple ways: Trust it to not inspect the traffic itself, trust it to not let itself be forced to reveal information and trust it to not be compromised by an attacker.

Tor expands on the relaying approach by using multiple proxies strung together. It is well established and considered a sturdy solution. It is restricted by the fact that it only supports TCP traffic and that it doesn't work reliably for real time traffic.

Table 3.1 shows the central aspects for each service.

Chapter 4

Analysis

The following chapter will start by describing a scenario that showcases the usage of the proposed system. Building upon that, the possible attackers that need to be considered are presented. The rest of the chapter then focuses on the requirements that the system needs to fulfill in order to meet its goals. The chapter is concluded with an examination of various ways of meeting these requirements.

4.1 Scenario

The following description uses terminology common in cryptographic literature. Alice and Bob represent arbitrary end users.

User Alice wants to exchange data with user Bob. Alice and Bob know each other.

They have previously exchanged cryptographic certificates issued by their privately operated certificate authorities. This is based on the requirement that every user runs his or her own certificate authority. The exact process of this certificate exchange can happen in a variety of ways (cf. [16]).

Additionally, Alice and Bob (and every other user of the system) are registered in a distributed hash table (DHT). A user can resolve the current IP address of a contact by querying the DHT for H(pubkey) (this is currently developed in a seperate thesis, cf. [7]).

The traffic exchange between Alice and Bob can for example be in the form of accessing a website, engaging in voice over IP communication or sharing access to a fileserver. In some cases, their roles are equivalent (e.g. VoIP), in others both of them have different roles (e.g. accessing a webserver). Alice and Bob want their exchange to be private. They also want to be able to use any of their devices for the exchange. While communicating

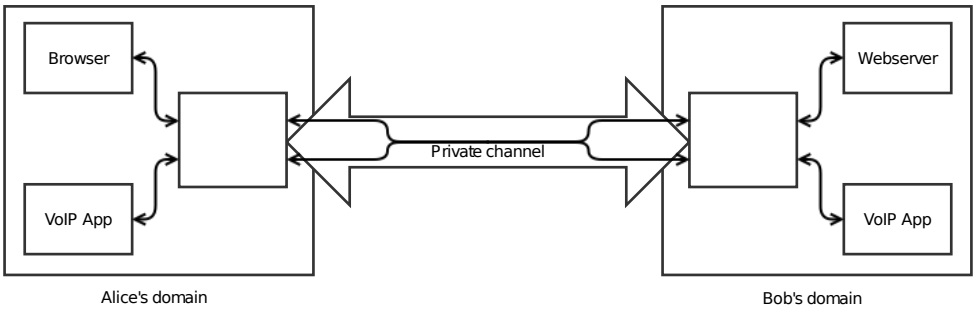

Figure 4.1: Abstract overview for the scenario

privately using the presented service, Alice and Bob also want to still be able to use the internet for unrelated activities in its conventional way.

Figure 4.1 shows an abstract overview of this setup. Some parts of the system shown in this figure are left blank - these are the parts that are going to be developed in this thesis. Questions that arise when confronted with this setup include: How can the VoIP- or Browser-traffic be intercepted? How can it be inspected? How can the system deduce that a connection is supposed to be made privately? How can users even address each other? And finally: How can the connection be equipped with privacy providing properties?

4.2 Attacker model

Alice and Bob want to exchange data privately, but *privacy* is not an absolute term. Building on the information from the earlier chapter about privacy, it is first necessary to define what kinds of privacy compromises the proposed system might have to face. How well the system fares with the different kinds of attackers will be described in chapter 7.

4.2.1 Passive attacks

In a passive attack, the attacker tries to compromise privacy without actively interfering with the data. A passive attacker tries to gather information by inspecting timings, packet sizes and other characteristics internet traffic can have. The attacker can also gain knowledge by inspecting information located in unencrypted packet headers. The power and vision of a passive attacker can vary widely. On the one end of the spectrum the attacker may only have access to traffic at one specific location. This might happen if there is a compromise of a local network. In this case, the attacker might be able to

intercept traffic by using techniques such as ARP spoofing. On the other end is the global observer, which is an attacker that is assumed to be able to inspect huge amounts of internet traffic simultaneously. A global observer has the ability to intercept traffic at various internet junctions.

4.2.2 Active attacks

An active attacker does not only inspect traffic, but also tries to break into the system. An active attacker may for example alter data in transit, introduce additional traffic into the system or replay traffic that was sent through the system. Same as for the passive attacker, the active attacker's power may vary widely. It is generally assumed that standard encryption technologies can not be broken even by the most powerful active attacker if used correctly. An active attacker might for example try to exploit flaws in the software by sending specially crafted packets. He or she might also try to gain information by introducing additional packets or altering valid ones and then observing the behavior of the system.

4.3 Requirements

The described scenario leads to certain requirements that need to be met. These can be split into functional and non-functional requirements.

4.3.1 Functional requirements

The following requirements need to be met in order to provide the basic functionality.

4.3.1.1 Traffic interception

The system needs to be able to intercept all traffic coming from a client. This is important because Alice and Bob should not be inherently prevented from using the service in any arbitrary setting. This does not mean that any arbitrary setting is supported from the start. Additionally, this makes it possible to be sure that no traffic, that should be routed through the privacy network, can accidentally leave the device in the clear. If all traffic is routed through the privacy providing device, all traffic can be dealt with adequately.

4.3.1.2 Traffic differentiation

Furthermore, the system needs to be able to inspect the intercepted traffic and react to it appropriately. It needs to be able to distinguish normal internet traffic from traffic that

should be provided with privacy properties. For normal traffic, Alice and Bob should be able to transparently continue using the conventional internet. For privacy traffic the system needs to distinguish between different application layer protocols and react to the supported protocols with the required functionality.

4.3.1.3 Application layer protocol extension

The system works by intercepting traffic and differentiating it based on the supported application layer protocols. Because of that, there needs to be a way to provide functionality that is tailored to a specific application layer protocol. This means providing a simple way of integrating support for new application layer protocols into the system.

4.3.1.4 Providing Privacy

Most importantly, a way that provides privacy to the traffic that is sent through the proposed system needs to be in place. This means that when Alice and Bob decide to engage in a private data exchange, this exchange should be equipped with certain privacy properties as described in the corresponding chapter. The process of starting a private data exchange should be as transparent as possible so that Alice and Bob are disturbed in their exchange as little as possible. Some kind of distinguishing property needs to be in place that indicates to the system that a data exchange is supposed to be done through the privacy network and that allows participants in the privacy network to easily start utilizing its features.

4.3.1.5 Addressability

As Bob needs to be able to provide services to other users like Alice, there needs to be a way in which Alice can reach Bob through the privacy network.

4.3.2 Non-functional requirements

These requirements may not be absolutely needed for the privacy network to function, but they are needed for it to function properly. This means guaranteeing a certain performance, being able to handle all kinds of traffic through modularity and being able to adjust to additional requirements that might come up in the future through extensibility.

4.3.2.1 Performance

The system developed in this thesis is supposed to be able to handle all kinds of traffic. One major use case is voice over IP traffic, which relies on low latencies. According to a recommendation by the International Telecommunication Union [32] one-way delays below 150ms are completely transparent to the user, while one-way delays of up to 280ms still satisfy users. These delays should not be exceeded.

4.3.2.2 Modularity

The different parts that constitute the system should be separated from each other as much as possible. The interaction between the parts should only happen through a small number of interfaces. This is desirable because it makes it possible to exchange parts of the system without impacting the rest of it.

4.3.2.3 Extensibility

It should also be possible to add additional features later on without needing to make changes to the system as a whole or even only parts of it.

Technological extensibility—The system should be designed in a way that does not inherently prevent certain kinds of traffic to be compatible with it.

Privacy related extensibility—Some promising features, like cover traffic, will not be implemented as a part of this thesis. The system should be able to handle the addition of such features, for example by providing generic interfaces.

4.4 Solution processes

The following section will present various possible approaches in regard to the different requirements. These approaches will be evaluated based on their usability for the proposed system.

4.4.1 Intercepting traffic

There are various ways to go about intercepting traffic.

One way would be to develop custom applications that provide services like VoIP or web browsing via a user interface. These applications could be part of the system and thus be easily connected to its privacy providing part. No real interception would be needed, as these applications could just pass on their data to other parts of the system. This approach has several drawbacks: The whole system would rely on the assumption that custom applications are developed for all use cases. This is a lot of work and restricts the ways in which the system can be used considerably. Additionally, users would need to use a seperate set of applications if they want to gain privacy for their communications. This introduces a usability barrier.

Another possibility would be using a proxy. Applications could connect to the proxy and send traffic to it. The proxy could then go about equipping this traffic with privacy providing properties. This setup would require applications to support this kind of proxy. They would need to be configured to use this proxy. Furthermore, one would rely on the application properly implementing the proxying functionality. Leaking of information that was supposed to be transferred privately could not be prohibited.

In order to circumvent the stated problems, the most reliable solution would be to try to intercept all traffic coming from a device. This would not necessitate any support by applications and would provide the possibility of preventing any leaking of private information.

4.4.2 Differentiating traffic

Depending on the way the traffic is intercepted, two different ways of differentiating traffic might be necessary: In order to decide if a packet is supposed to be transferred privately or not and in order to decide which application a packet belongs to.

This is no problem when using dedicated applications. All traffic coming from them should be transferred privately and they can communicate with the system in order to make packets assignable.

With a proxy, users connect their applications to it when they want to use the privacy services. Because of this, it can also be assumed that all incoming traffic is supposed to be transferred privately. In this case, it would only be necessary to differentiate different application layer protocols and react accordingly to each of them.

When intercepting all traffic coming from a device, it is necessary to both differentiate normal internet traffic from traffic that is supposed to be transferred privately and to differentiate different application layer protocols.

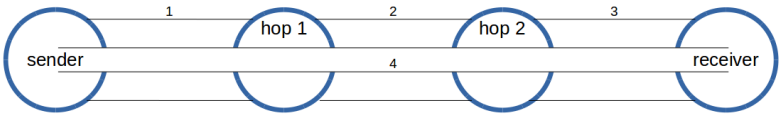

Figure 4.2: Bamboo-like tunnel

4.4.3 Providing privacy

As shown in chapter three, there are two generally different ways that can be used to gain privacy for traffic: Proxying and Multicasting/Broadcasting.

Broadcasting runs into issues when confronted with the requirements imposed by our system: Supporting VoIP is a central goal. This would mean that all VoIP traffic is sent to all participants at any time. Depending on the size of the network and the computing power (= hardware) assigned to the privacy providing system by individual users, participants could be easily overwhelmed by this amount of traffic. Additionally, all traffic would need to be encrypted, leading to all participants needing to try to decrypt every single VoIP packet sent through the network.

Dividing the network into separate sections, as Bitmessage proposes, might partly mitigate this problem. This puts an upper bound on the privacy provided by all other participants that might possibly be the intended recipient. Furthermore, as the system proposed in this thesis is supposed to run on devices freely chosen by the user, it would also be hard to find an upper bound for network segment sizes that allows all users to process all packets.

In regard to proxying, chapter 3 already introduced, that an approach using chained proxies is promising. In this case, the questions of how proxies are informed about the next hop and how the encryption is built, remain.

One approach would be to inform each proxy about the next proxy it is supposed to connect to. The proxies can then establish secure connections with each other until there are secure connections all the way through to the target host. Through all these connections, both endpoints could establish another secure connection in order to avoid relaying proxies to be able to understand the content. This results in a bamboo-like tunnel structure as can be seen in figure 4.2.

In this setup, each relaying proxy would need to decrypt the traffic once, and encrypt it once again before sending it to the next hop. Sender and receiver would additionally need to encrypt and decrypt once more respectively.

The problem with this approach lies in the question how the intermediary proxies are informed about the proxy they are supposed to connect to. Informing each proxy seperately completely undermines the privacy providing properties, as an observer

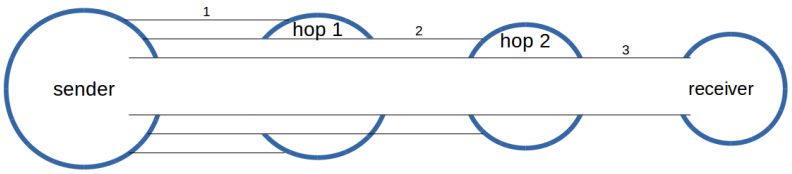

Figure 4.3: Telescope-like tunnel

could easily figure out which proxies are going to be used.

Letting the proxies relay that information to each other in an unencrypted way through already established secure connections on the other hand would lead to relaying proxies knowing about other relaying proxies that are being used.

The best option therefore is to encrypt this information at the sender for every relaying proxy and then let the information be passed down the tunnel. This would lead to secure connections being established between all involved proxies as well as some kind of cryptographic exchange between the sender and every relaying proxy.

In order to cut down on this overhead, an improvement to this approach would be to directly establish nested secure connections between the sender and every relaying proxy. This results in a telescope-like tunnel structure (c.f. [4]) as can be seen in figure 4.3.

Here, every relaying proxy only needs to decrypt once and then simply relay the remaining data. Here, the sender has the bulk of the workload by having to encrypt the data multiple times, once for every hop.

This has the desirable side effect of reducing the amount of cryptographic operations a relaying proxy has to do after a secure connection has been established to only decrypting once.

4.4.4 Addressing users

In order to be able to establish a private connection to another user, certain information is necessary. At the very least, the IP address is needed. Because the connections that are established in this system need to be secure, cryptographic material like public keys is another important thing to know.

A system, that provides these features and allows addressing other users in a way, that does not compromise privacy, is developed in a seperate thesis [7]. A detailed description of this system can be found there.

Chapter 5

Design

In order to provide a service that matches the needs and fulfills the requirements outlined in the previous chapter, a framework was designed. This framework is called *PrivacyBox* and can run on a designated device or even on the client.

5.1 Overview

Chapter four established the basic requirements that need to be met. In order to fulfill them, several components need to be developed. This includes a way to intercept all traffic leaving a device, the ability to differentiate traffic in regard to privacy requirements as well as the application layer protocol that it belongs to. Furthermore, there needs to be functionality allowing the establishment of secure connections between users as well as a way of addressing these users. Last but not least, there needs to be a part that provides the relaying functionality. Figure 5.1 is a further developed version of figure 4.1. It now shows how the different modules interact.

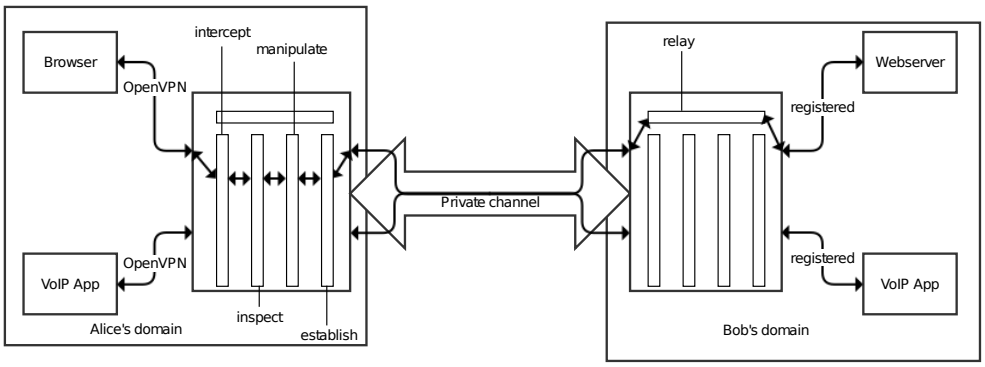

Figure 5.1: Scenario with module interaction

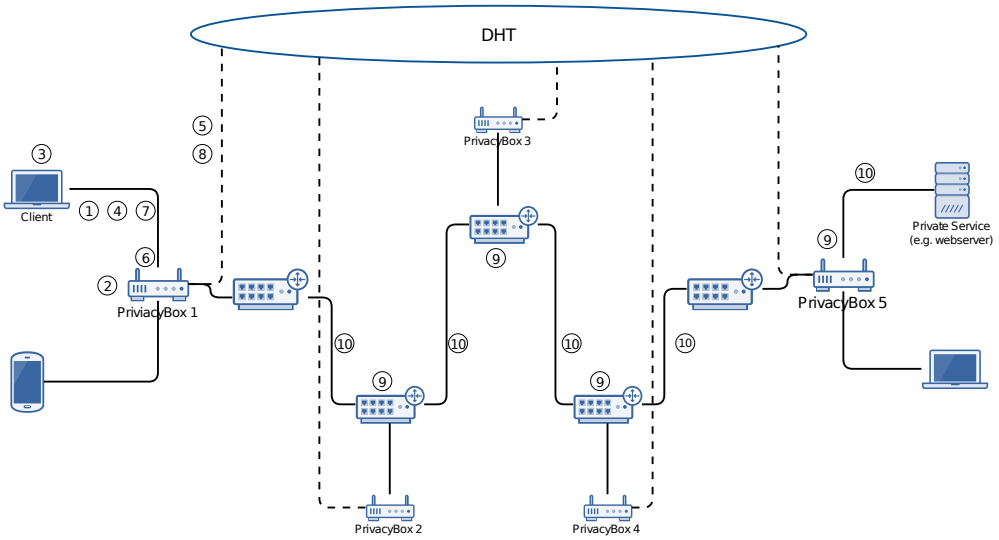

Figure 5.2: Overview over an exemplary setup

As an introduction to the design of this system, an overview over an exemplary setup will be described. While figure 5.1 provides a view of the various modules of the PrivacyBox-Framework, figure 5.2 focuses on how a tunnel is established.

The pictured Privacy-Network has various participants. There are five devices that run the PrivacyBox-Framework and thus participate in the Privacy-Network: PrivacyBoxes 1 through 5. Additionally, there are two devices connected to PrivacyBox 1 and PrivacyBox 5 respectively. They are either clients accessing the PrivacyBox network through their corresponding PrivacyBox or a service providing for example a web server to the PrivacyNetwork. Clients and services may be running the PrivacyBox-Framework themselves or - as in this case - may be connected to a designated PrivacyBox. This connection may be established via WiFi, OpenVPN or any other way that allows relaying all traffic a client sends and receives through the PrivacyBox. All PrivacyBoxes in this example are located in small networks behind a router. One example for a setup like this is when a user runs his or her PrivacyBox in a home network.

Additionally, the PrivacyBoxes use a distributed hash table that stores the information needed in order to establish connections (cf. [7]). The diagram shows a situation where a client connected to PrivacyBox 1 wants to access a service connected to PrivacyBox 5. In order to show the interaction of the different participants, an exemplary use case is provided. The locations of the numbers in Figure 5.2 correspond to the places where the events in the following enumeration occur:

1. A user connects to the PrivacyBox-Framework running on PrivacyBox 1. All traffic from this moment on is relayed through the framework.

2. As all traffic is now relayed through the PrivacyBox-Framework, it is able to inspect the traffic and determine its destination.
3. The user opens a browser and enters the address of a website that one of his or her contacts is running inside the privacy network. The address ends with the TLD *.privacy*.
4. A DNS request is sent by the users device. This request is intercepted by the PrivacyBox.
5. The PrivacyBox determines that the request is targeting a *.privacy* TLD. It accesses the DHT and requests the contact's address corresponding to the domain name.
6. It then creates and sends a DNS reply providing this address. It also creates a session, remembering that HTTP traffic directed at this address should be sent through the privacy network.
7. As soon as the client receives the reply, an HTTP request is sent to the previously received address.
8. PrivacyBox 1 intercepts this HTTP request. It queries the DHT again, this time requesting information about three random relaying PrivacyBoxes.
9. Once PrivacyBox 1 has gathered this information, it can start establishing a tunnel through the received relaying PrivacyBoxes.
10. When the tunnel is established, the HTTP request is sent through it, finally being relayed by PrivacyBox 5 to the private service. The response is sent through the same tunnel, only in the other direction. All further HTTP requests to that service are directly sent through the tunnel.

5.2 System design

In general, the framework is designed to be as transparent to the user as possible. In order to achieve this, the framework intercepts all traffic received from a client connected to or running a PrivacyBox. To differentiate standard traffic from traffic that needs to be relayed through the Privacy-network, the TLD *.privacy* is used. Traffic that is recognized as such needs to be dealt with in an appropriate way.

The functionality of the PrivacyBox-Framework is split up into different modules. Each module has one central task. The modules that are part of the current design are:

- Traffic interception module
- Traffic inspection module
- Application layer protocol module(s)

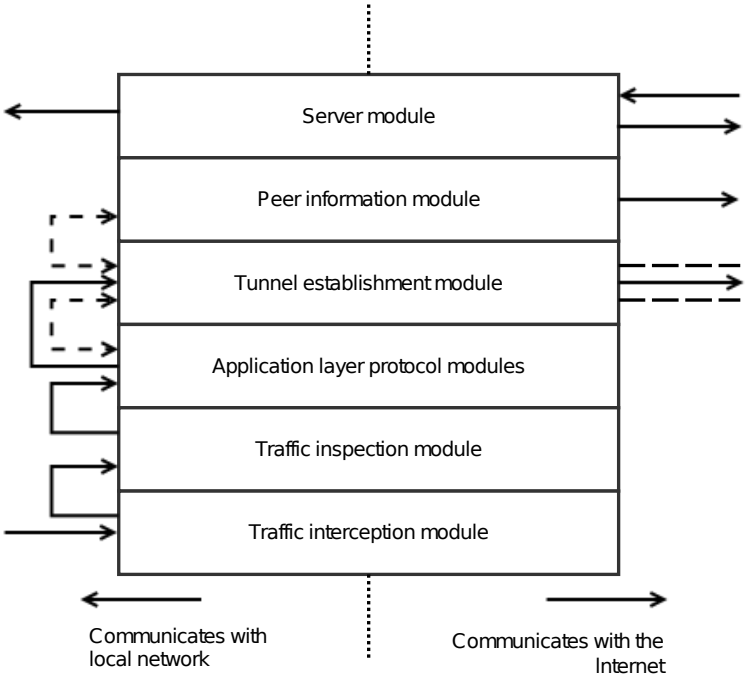

Figure 5.3: PrivacyBox modules

- Tunnel establishment module
- Peer information module
- Server module

These modules represent what is needed to achieve a basic functionality. Their basic interaction can be seen in Figure 5.3. It will be explained in the following sections.

In order to completely fulfill all of the goals, additional modules are needed. This will be elaborated in chapter 7.

In the following, an overview over each of these modules will be given.

5.2.1 Traffic interception module

The traffic interception module needs to be able to receive all network traffic that leaves a device. This is important in order to prevent the possibility of traffic leaking and thus potentially conveying privacy-related information to an observer.

If the PrivacyBox-Framework is running on the device that wants to use it itself, the traffic can easily be intercepted there. If the PrivacyBox-Framework runs on a device different from the client device used to access the privacy network, additional measures

need to be taken in order to redirect all traffic. For the prototype, OpenVPN was used to connect a remote client to the PrivacyBox-Framework.

5.2.2 Traffic inspection module

The traffic inspection module has the task of loading the available application layer modules and passing the packets on to them.

An application layer protocol module provides functionality specifically taylored to provide the ability to create private connections for one particular application layer protocol.

For both UDP and TCP traffic, each available application layer protocol module is called until either a protocol module takes responsibility for the packet processing or none of the protocols matches. If no matching protocol is found, there is no way to provide privacy to the packet and it is simply forwarded as is. If an application layer protocol module takes responsibility, further processing is delegated to it.

5.2.3 Application layer protocol modules

The task of an application layer protocol module is to detect application layer traffic it is responsible for, check if this traffic should be routed through the privacy network and – if so – initiate the necessary steps.

As a different application layer protocol module has to be provided for every application layer protocol, there can not be one universal design. Apart from the previously mentioned tasks, an application layer protocol module can also implement functionality for tunnel-events such as the establishment or shutdown of a tunnel it initiated.

An exemplary application layer protocol module, that is connected to the goal of providing privacy to low latency traffic, is the VoIP-module. It is currently developed in a seperate thesis (cf. [6]).

5.2.4 Tunnel establishment module

The tunnel establishment module is responsible for building the tunnels through the privacy network. It consists of two parts: a management module and a tunnel module. The management module manages the tunnels. By using the tunnel module, it can create various nested tunnels. Figure 5.4 shows the part of the tunnel establishment that happens on the initiating PrivacyBox.

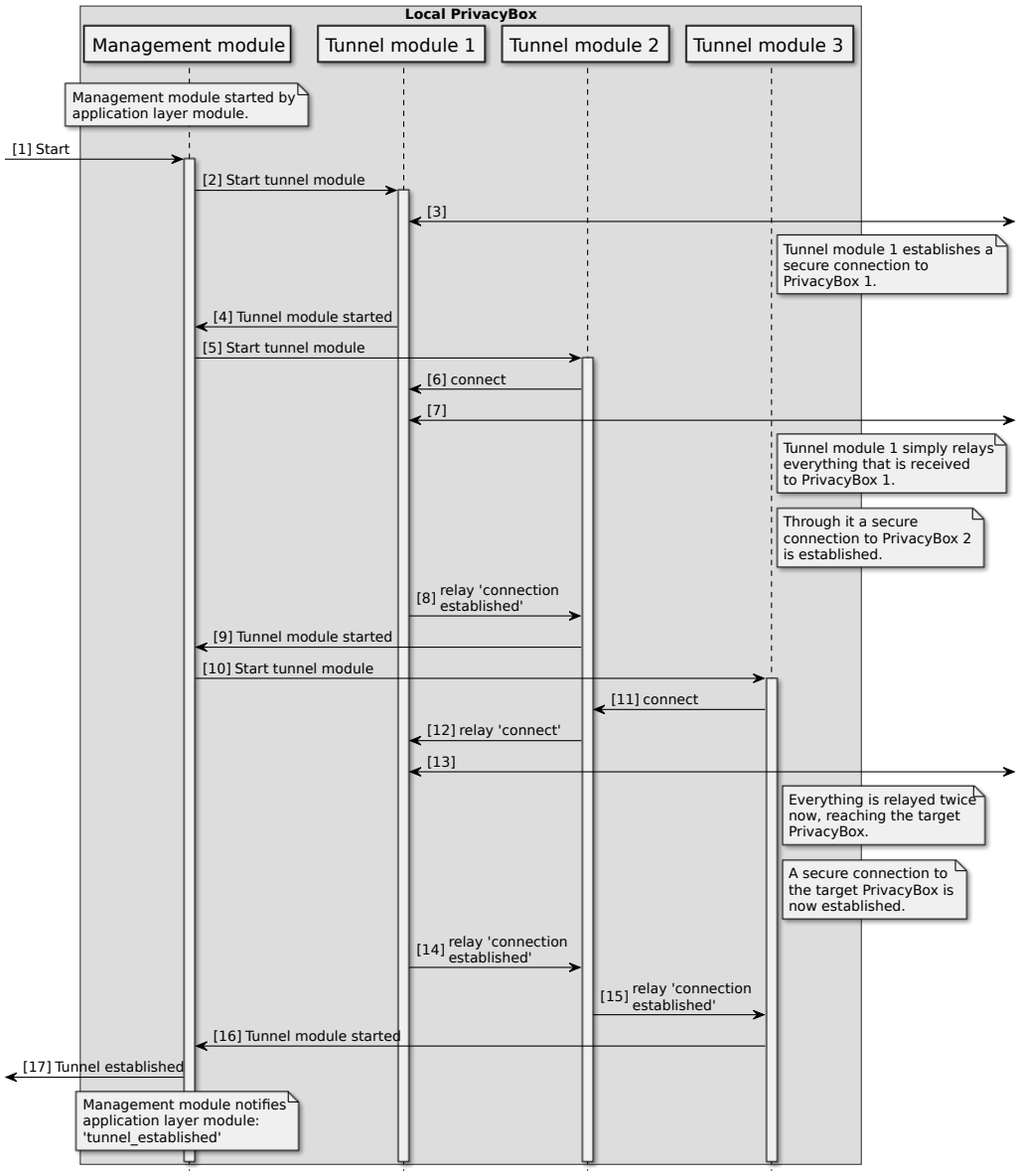

Figure 5.4: Tunnel management module

5.2.5 Peer information module

This module is the interface to the DHT. It allows requesting the contact information of a known PrivacyBox as well as querying for random PrivacyBox addresses that can be used as intermediary relay nodes.

The functionality that is needed for all this to work properly, is developed in a seperate thesis [7]. A detailed description of how this part works can be found there.

5.2.6 Server module

Finally, the server module is there to provide the relaying functionality. It is largely independent of the rest of the system.

This module simply listens for TCP and UDP connections on a designated privacy-port. If installed in a home network or similar setting, port forwarding needs to be configured on the router in order for packets to reach the PrivacyBox. If a packet is received, the server module waits for connection instructions from the initiating host. It fulfills these instructions and from then on simply relays everything it receives from both directions.

The server module is responsible both for relaying data to other PrivacyBoxes as well as relaying data to hosts in the local network. If the connection instructions request a connection to a remote PrivacyBox, it serves this request by establishing the connection and then acting as described in the follwing chapter. If a host in the local network is supposed to be accessed, the procedure is slightly different.

Services in the local network, that are supposed to be accessible through the privacy network, need to inform the PrivacyBox-Framework about their existence. This means mapping a port on the PrivacyBox to the IP address of the host on which the service is running.

A user who wants to access a service run by one of his or her contacts needs to know about the existence of the service as well as the port on which it is running. A request that is supposed to ask for a connection to a local service requests a connection to the same PrivacyBox that received the connection request but for a connection to the port on which the service subscribed.

For example, if a user wants to provide a website inside the privacy network, he or she might run it on a device in the local network with IP address *192.168.10.10*. In order for the server to be accessible, the PrivacyBox-Framework running on another device in the local network would need to be informed about this webserver, by entering the mapping *port 80 -> 192.168.10.10* in a configuration file. If the PrivacyBox then receives a connection request to port 80 of its own IP address (basically requesting to establish a

connection to itself), it understands that this request is directed at a local service. It can then look up the mapping and establish a connection to the local service.

This results in a situation where there can't be multiple private services running on different devices and using the same port in the local network. The reason for this is, that the port, a request is directed to, is mapped to one IP:port combination.

5.3 Security

Traffic sent through the privacy network needs to be confidential. This is something inherently expected from a privacy providing system. It is also of prime importance, because packets are relayed through the PrivacyBoxes of other unknown users. At every step, the information that can be gained from a packet needs to be as constrained as possible.

How confidentiality is achieved and encryption applied is also crucial, because even encrypted traffic might provide information to a passively observing attacker. This would be the case if packets were only encrypted once end-to-end and then simply relayed by the intermediate relays. In this case, the observer might be able to observe the same encrypted payload at different stages of the tunnel and thus circumvent the privacy provided by relaying entirely.

If packets are encrypted multiple times, an attacker might be able to deduce information about the connection by inspecting the change of packet sizes at every hop. As every PrivacyBox removes one layer of encryption in this case, the packet gets smaller and smaller by a set amount of bytes. This problem can be prevented by introducing a padding scheme in order to achieve constant packet length. Such a scheme is not yet part of the presented system though and would need to be developed in another thesis.

Section 5.2.4 already explained how the tunnels are established. The PrivacyBox-Framework currently works with TCP and UDP traffic. TLS and DTLS are used to provide a secure connection to these protoclols. In order to avoid the possibility of correlating traffic at different stages of the tunnel, encryption is applied in the following way:

1. The local PrivacyBox (PB0) establishes a TLS/DTLS connection to the first hop (PB1).
2. PB0 confidentially tells PB1 about the second hop (PB2) through that connection.
3. PB1 subsequently establishes a connection to PB2 (in case of TCP/TLS) and prepares to relay everything received from PB0 to PB2.
4. PB0 then establishes another secure connection with PB2 through the secure connection with PB1.

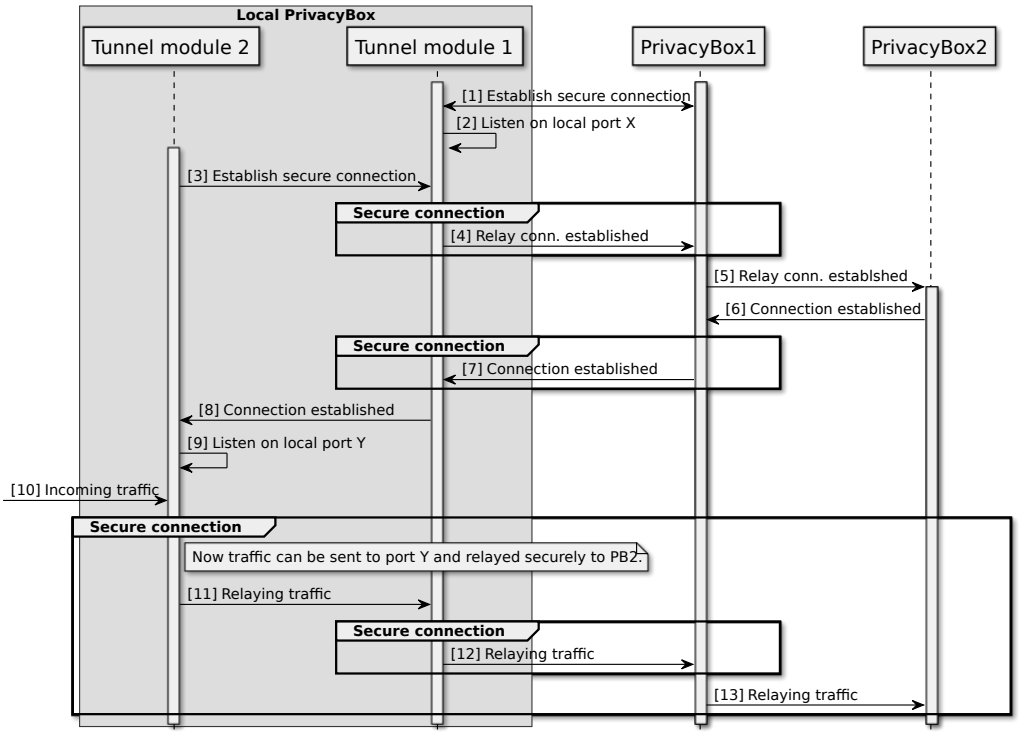

Figure 5.5: Client tunnel establishment

5. This process is repeated until a secure connection between PB0 and the intended target PrivacyBox is established.

The resulting connection is encrypted end-to-end and enclosed by multiple layers of encryption. As each intermediate privacy box applies a decryption function on the packet, the payload differs at each step of the tunnel. Figure 5.5 shows this process. In this case, only one relaying PrivacyBox is shown for reasons of simplicity.

Chapter 6

Implementation

The implemented framework builds on the existing technologies presented in chapter 2 and uses them in order to reach its goals. This chapter will elaborate on the implementation details of the different framework modules.

The architecture of the PrivacyBox-Framework was described in the previous chapter. In the following, the different modules and their functionality are explained in detail. The implementation was done in Python.

6.1 Traffic interception

The traffic interception module enters specific rules into the firewall that allow modules further down the line to inspect and modify packets. If the PrivacyBox-Framework is running on the samme device that is connecting to the privacy network, these rules forward all traffic leaving the device into userspace by using libnetfilter_queue. If it is running on a designated device on the other hand, these rules forward all traffic being forwarded by the device into userspace.

The rules for forwarding all traffic being forwarded (i.e. belonging to a remote PrivacyBox-setting) look like this:

```
iptables -A FORWARD -j NFQUEUE -p udp
iptables -A FORWARD -j NFQUEUE -p tcp
```

In order to be able to inspect the packets added to the queue, the interception module also opens the socket in userspace. Additionally, it sets the method that should be called for every packet received by this socket. This socket is handled by the reactor.

Finally, as the interception module is the most basic of the modules, it also handles certain setup and shutdown tasks. It starts the reactor once the iptables rules are applied

and the socket added to the reactor. It also stops the reactor and removes the iptables rules if the PrivacyBox-Framework is shut down.

6.2 Traffic inspection

The traffic inspection module is instantiated by the traffic interception module. During instantiation, it loads all modules listed in a configuration file. This allows additional protocols to be easily supported without necessitating changes in parts of the framework.

In order to be loaded, the application layer protocol modules need to follow a certain naming scheme: This entails providing a module name in the config (e.g. *dns*) and consequently naming the class by prepending *module_* to the name added to the config (e.g. *module_dns.py*)

As soon as the reactor is started by the traffic interception module, all packets leaving or being relayed by the device on which the PrivacyBox is running are forwarded to the traffic inspection module. As currently only UDP and TCP protocols are supported, the inspection module checks for these transport layer protocols and passes the packets to the corresponding protocol modules if appropriate.

As the traffic inspection module is the central packet processing point before delegating to specific application layer protocol modules, it has the ability to act as an intermediary for different application layer protocol modules if needed. For this, it also provides a notify method that can be called by application layer protocol modules and can transfer messages between them.

6.3 Application layer protocol modules

As already mentioned in the design chapter, application layer protocol traffic is not intercepted by one specific module. Instead, one specific application layer protocol module is needed for every application layer protocol.

6.3.1 Abstract overview

Every application layer protocol module has to implement three methods:

- **process(self,pkt,payload)**: Called by the inspection module. Decides what to do with the payload, depending on the content of the packet.
- **tunnel_established(self,pkt,payload,last_port)**: Called by the tunnel-modules once a tunnel is established. Implementation depending on the application layer

protocol module. Most likely needs to add iptables rules that forward traffic through the created tunnel. Also needs to decide, what to do with the first intercepted packet.

- **tunnel_closed(self,last_port)**: Called by the tunnel-modules once a tunnel is closed. Implementation dependent on the application layer protocol module. Most likely needs to remove iptables rules.

These three methods allow the modules to initiate tunnel establishment (which is one of their most likely tasks) and react accordingly if the state of the tunnel changes.

6.3.2 Exemplary module: HTTP

An exemplary module, that was developed for this thesis, is the HTTP-module. This module actually consists of two modules: A DNS module and an HTTP module.

The DNS-module is a UDP module and is called for UDP traffic by the inspect module. Its first task is to check if the UDP payload is indeed a DNS packet. This is done by looking at the destination port. If the port is 53, the packet will be further processed. If not, the UDP module simply returns and the packet will be accepted unmodified.

Once a packet is recognized as a DNS request, the DNS module inspects the content of the request. The relevant part of the DNS request here is the name of the queried domain. As the top level domain *.privacy* is used to signify that traffic is supposed to be sent through the privacy network, the module checks if the queried domain ends with this TLD. If this isn't the case, it just returns and the packet is relayed normally. If a DNS request is recognized as directed at a *.privacy* domain, the DNS module creates a DNS reply.

For this, it queries the peer information module (which is described in its own thesis [7]) for the IP address of the destination PrivacyBox. As this module was not yet finished at the time the work on this implementation was done, a dummy peer information module was created that returned certain IP addresses. With this information the DNS module can create the DNS response containing the resolved IP address and send it to the client. The DNS module also uses the notification functionality of the intercept module in order to tell the HTTP module, that all HTTP requests sent to the just resolved IP address need to be tunnelled.

As soon as the DNS response is received by the user's device, an HTTP request is sent. This is intercepted by the HTTP module based on the information received from the DNS module. The HTTP module then queries the peer module for a number of random intermediary PrivacyBoxes.

It then starts the module responsible for tunnel establishment and passes the queried IPs as well as the *tunnel_established* and *tunnel_closed* functions to it.

The *tunnel_established* function is called by the tunnel establishment module once the tunnel is done. It gets passed the local port on which the just created tunnel is listening as well as the packet that initiated the tunnel. It also creates an iptables rule that redirects all TCP traffic directed at port 80 of the resolved target IP address to the local port on which the tunnel is listening. Thus all traffic directed at the target privacy box will now be sent through this tunnel.

6.4 Traffic relaying

The final module that needs to be presented is the module responsible for establishing the tunnels. It consists of two parts: a management module and a tunnel module.

6.4.1 Management module

The management module starts the tunnel modules, whose task it is to establish the tunnels. As previously mentioned, the management module is called when an application layer protocol module needs to establish a tunnel. It is passed the IP addresses of the PrivacyBoxes that are used for relaying, as well as the IP address of the target PrivacyBox. Additionally, it gets passed the *tunnel_established* and *tunnel_closed* functions of the application layer protocol module.

It uses this information to start the tunnel modules in the right order and eventually notify the application layer protocol module that a tunnel has been established. Figure 5.4 shows the processes of the tunnel establishment that are linked to the management module.

6.4.2 Tunnel module functionality

The tunnel modules themselves handle the basic connection establishment (in case of TCP), the establishment of a secure connection (TLS/DTLS) and finally the relaying.

Each tunnel module first connects to another PrivacyBox that is supposed to act as a relay and then tells this PrivacyBox about the next hop. In order to be able to do this, each tunnel module is passed two IP addresses by the management module: The first IP address is the address of the PrivacyBox it is supposed to directly connect to. The second IP address is the address of the PrivacyBox to which the traffic is supposed to be relayed. The tunnel module uses the first IP address to connect to the first PrivacyBox. Before relaying information is sent, a secure connection is established. In case of TCP traffic, TLS is used and in case of UDP traffic DTLS is used. Once the establishment of the secure connection is done, the PrivacyBox is notified about the next hop in the tunnel.

For this, a SOCKS-like protocol is used. This protocol essentially tells the PrivacyBox about the IP address of the next hop.

Once this procedure is finished, the tunnel module starts a local server. This server listens on a random local port. The tunnel module then calls the management module's *tunnel_established* function, thus conveying both the port and the fact that the tunnel has been established.

This description holds true for the first tunnel module instance that establishes the tunnel to the first PrivacyBox. The procedure is slightly different for the following tunnel module instances. The difference here is that they do not first establish a connection to the remote PrivacyBox. Instead, they connect to the local port of the previously created tunnel module instance. As all traffic that is sent to that port is relayed to the remote PrivacyBox next in line, this seems to the newly created tunnel module as if a connection to this PrivacyBox was established.

This sequence is used in order to gain multiple layers of encryption. The first layer of encryption is provided by the secure connection between Tunnel module 1 and PrivacyBox 1. The second layer is provided by the secure connection between Tunnel module 2 and PrivacyBox 2 and is established through the first secure connection. This process can be repeated as often as needed.

Because of this, no PrivacyBox can read meaningful information that it doesn't need. Above all, a relaying PrivacyBox can only read the contents of the SOCKS-like protocol that it itself needs. Thus, no relaying PrivacyBox knows more about the tunnel than the previous and the next hop.

6.5 Server module

The last part of the PrivacyBox-Framework is the server module. As every PrivacyBox acts as an entry point into the privacy network for connected clients but also as a relay node for other PrivacyBoxes, it needs to be reachable by other PrivacyBoxes.

The server module registers two servers, one for TCP/TLS connections and one for UDP/DTLS connections. Both listen on the same port. This port is the same for every PrivacyBox.

When another PrivacyBox connects, a TLS or DTLS handshake is initiated first. After that, a SOCKS-like message is sent by the other PrivacyBox. Using the information contained in this message (particularly the IP address of the next hop), the server initiates a connection to the next hop (for TCP) or saves this information until additional data is received (in case of UDP). Once this is done, the server simply relays everything in both directions.

Establishing a connection to a local service is not much different from the described procedure. The difference is, that the SOCKS-like message requests a connection to the same PrivacyBox that receives it. This tells the PrivacyBox, that a connection to a local service is supposed to be established. The PrivacyBox then looks up the IP address of the service in a configuration file. It uses this information to establish a connection to the local service (or save the information until additional data is received in the case of UDP), just like it would do when connecting to another PrivacyBox.

Chapter 7

Evaluation

In order to make a clear statement regarding how well the proposed system is able to fulfill the requirements laid upon it, evaluation of certain metrics is needed.

This evaluation will focus on two different aspects: The performance of the system on the one hand and the extent to which security and privacy properties are provided on the other.

7.1 Performance

The performance of the system is of prime importance. A central goal of this system is providing a low-latency anonymity network that makes anonymous and private voice over IP communication possible. Because of this, the latency of the system needs to be evaluated. Additionally, the overhead regarding packet sizes introduced by the layered encryption will be examined. The system should be able to run in a private setting where download but especially upload speed might be highly limited. A large overhead might limit relaying capabilities.

7.1.1 Latency

According to the International Telecommunication Union [32], a one-way delay of up to 280ms satisfies users. This can be seen in figure 7.1. Therefore, this number will be taken as the upper bound for acceptable one-way delay.

7.1.1.1 Tunnel length

In order to understand the delays present in the proposed system, measurements were made in a local setting using the prototype implementation (unless noted otherwise):

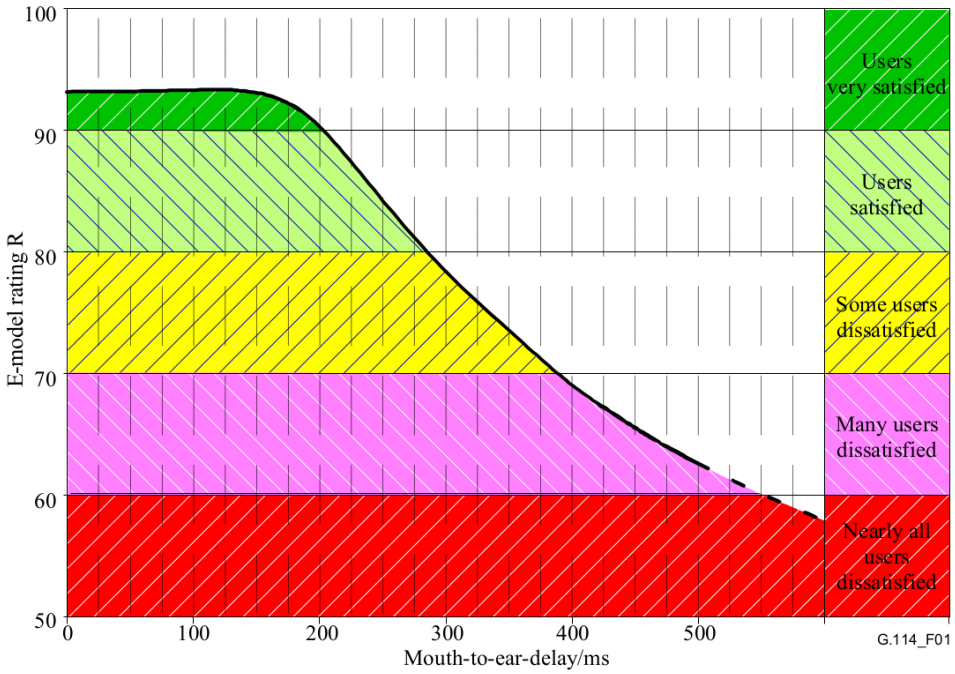

Figure 7.1: Effects of delay

A Raspberry Pi 1 Model B [33] was used to host multiple instances of PrivacyBox servers. Additionally, a custom echo-server was hosted on the Raspberry Pi in order to allow measurements. The PrivacyBox-Framework was also installed on a personal computer that directly connected to the fake privacy-network represented by the Raspberry Pi.

As all relaying PrivacyBox instances were running on the same device, the differentiation between them was made based on ports. There were one to six instances running at a time.

In order to infer conclusions about a real world setting from this setup, some preliminary considerations have to be taken into account:

- Measurements were made only once a connection had been established.
- Round-trip latencies between the client PC and the Raspberry Pi as well as between the various PrivacyBox-Instances running on the RaspberryPi were lower than 1 ms. This is a negligible amount and will be ignored in the following calculations.
- Real world round-trip latencies vary greatly depending on the distance between hosts. Based on measurements by Verizon [34], an average intracontinental (inside one continent) round-trip latency of 38ms and an average intercontinental (between different continents) round-trip latency of 90ms will be assumed.

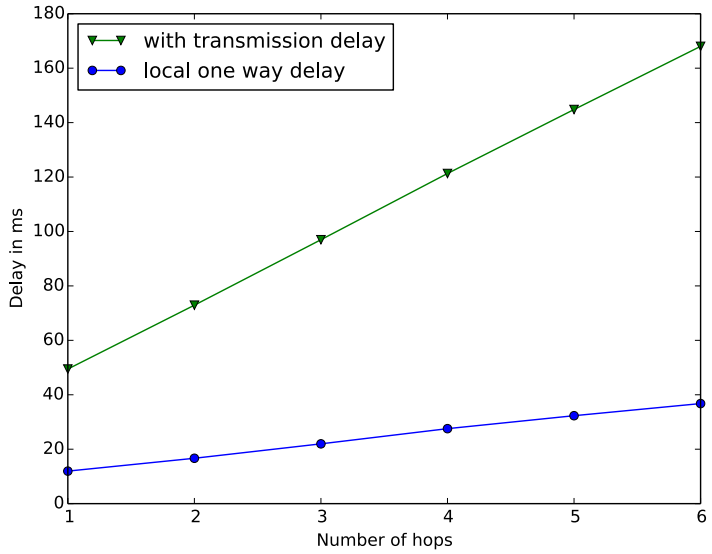

Figure 7.2: Intracontinental one-way delay for TCP traffic

- These latencies need to be added to the measured latencies. For this one has to calculate the one-way delay for intra- and intercontinental delays by deviding the round-trip delays by two. This one way delay then has to be added to the measured latencies in the following way: (one way delay * (Number of hops + 1)).

Measurements were made for both TCP and UDP traffic. The results for TCP can be seen in Figure 7.2 and Figure 7.3. The results for UDP can be seen in Figure 7.4 and Figure 7.5.

Comparing the figures for TCP and UDP, the most surprising detail seems to be that TCP is much faster than UDP (e.g. one Hop TCP RTT: 24ms; one Hop UDP RTT: 53ms). The reason for this can be attributed tto two factors:

1. Implementation: The twisted framework natively supports TLS connections, but doesn't support DTLS. Because of this, DTLS had to be implemented in a non-standard way for the prototype. This meant lower level network programming independently of the rest of the framework. As this part of the framework isn't as optimized as the rest of it, the slow processing speed can be attributed to it.

2. Local setting: The measurements were taken in a local setting, where all participants were connected via ethernet in a small local network. In this setting, the disadvantages of TCP for real time communication do not come in to play. There is virtually no packet loss, no congestion etc. Because of this, advantages of UDP compared to TCP can not be observed.

Apart from this, the numbers show that the amount of delay, introduced solely by

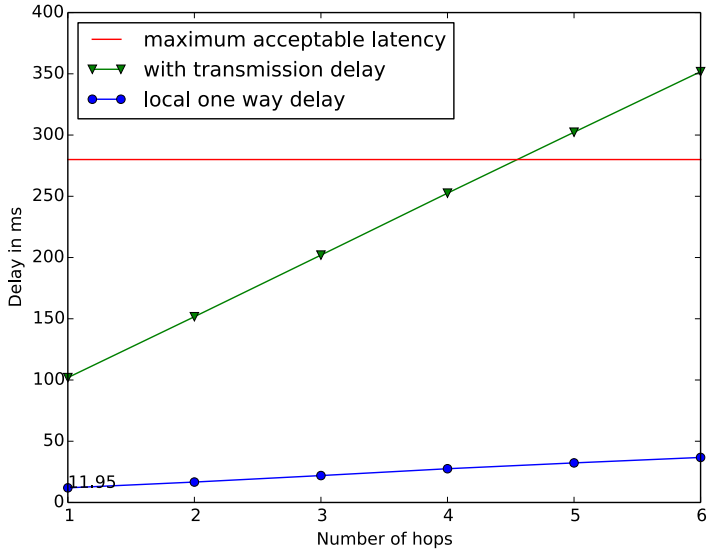

Figure 7.3: Intercontinental one-way delay for TCP traffic

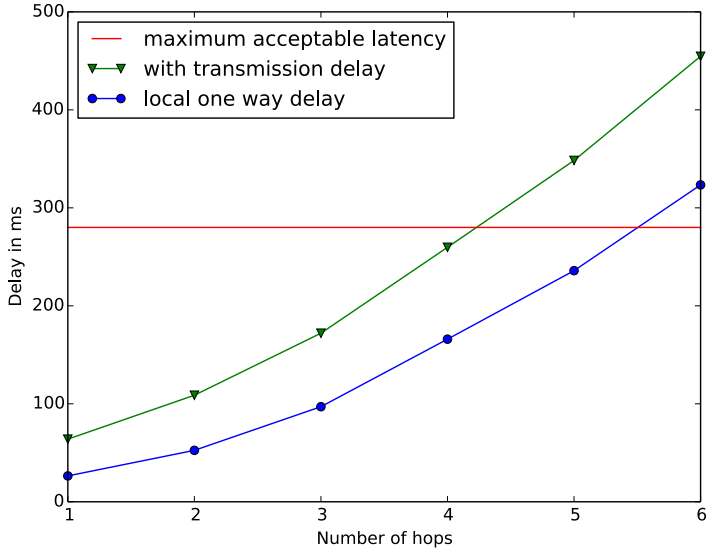

Figure 7.4: Intercontinental one-way delay for UDP traffic

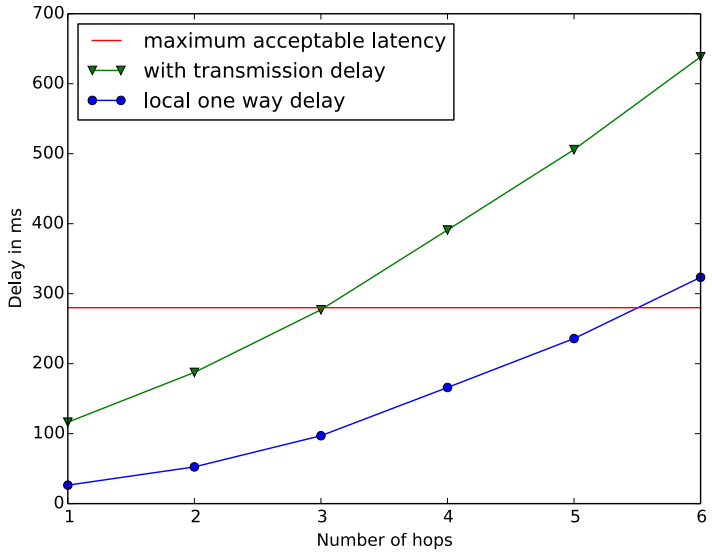

Figure 7.5: Intracontinental one-way delay for UDP traffic

processing traffic through six hops, introduces a one way delay of 37ms for TCP. This is roughly 6ms per hop. Compared to the time required for the traffic to travel from hop to hop, this is a very small delay. A completely intracontinental tunnel (19ms one way transport delay per hop) would satisfy users even when using as many as six relay nodes. A completely intercontinental tunnel (45ms one way transport delay per hop) could go as high as four relay nodes and still satisfy users.

The numbers look somewhat less promising for UDP. A maximum of three relay nodes is satisfying for users both for intra- and intercontinental tunnels. As mentioned, this mostly traces back to implementation issues.

Of course, these numbers might be distorted due to the local setting. There is practically no packet loss, no network congestion and no packets arriving out of order. This means that the numbers for TCP might be worse in a real world setting, especially for intercontinental connections. As UDP doesn't get slowed down because of these incidences, UDP numbers should more or less stay the same. Of course, transferring packets through a congested network will always be slower than in the local setting used for these measurements. Tests with a connection to a remote intracontinental server did not show drops in latency. This can be interpreted as evidence for the stability of intracontinental connections.

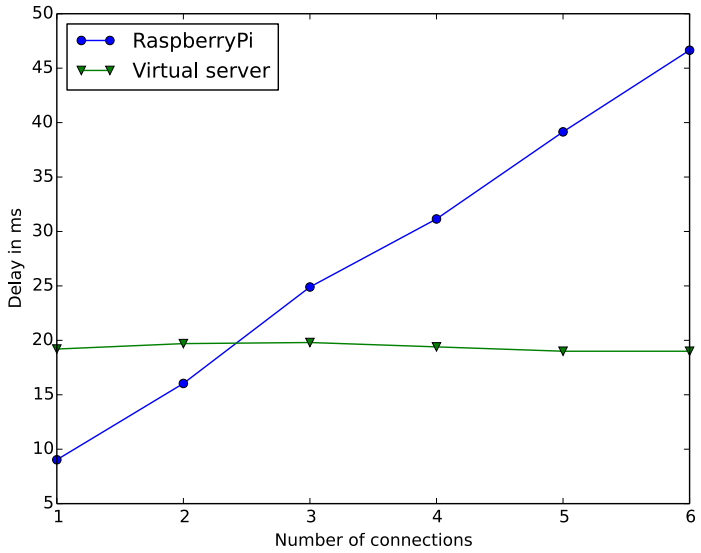

Figure 7.6: One-way delay for simultaneous connections

7.1.1.2 Simultaneous connections

Apart from the number of hops, the number of simultaneous connections, that need to be relayed by a PrivacyBox, can also have an effect on latencies. This is influenced strongly by the hardware on which the PrivacyBox-Framework is running. In order to observe this, up to six simultaneous connections were made both to a RaspberriPi running in the local network and to a lower grade virtual server hosted in Berlin. Figure 7.6 shows the results. The RaspberryPi gets overwhelmed pretty fast, having more than 45ms one-way delay per connection for six simultaneous connections. The virtual server on the other hand can handle six simultaneous connections without a problem, providing no real perceivable increase in delay.

7.1.2 Packet sizes

Another property of the proposed system, that is relevant for evaluating its usability, is the increase of packet sizes. As a completely decentralized approach is being taken, users may need to be able to support the amount of up- and download traffic imposed by the PrivacyBox on their home connection.

Figure 7.7 shows the size of an originally 160 byte packet encrypted as often as required by the hops used for the connection. A packet has to be encrypted (n + 1) times, with n being the number of hops. If there is only one hop, for example, the payload is encrypted twice: Once for the relaying PrivacyBox and once for the target PrivacyBox.

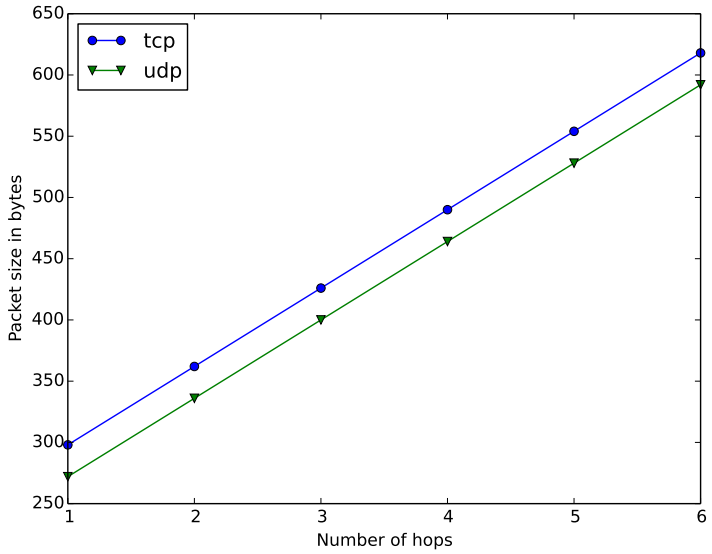

Figure 7.7: Packet sizes

As can be seen in the figure, the overhead is roughly the same for UDP and TCP, with UDP having slightly smaller packet sizes. The values increase by exactly 64 bytes per hop. This is no surprise, as the same encryption algorithm is used at each hop. This overhead is the result of a TLS header, a MAC and padding. As a result, it is not dependent on packet size.

For a PrivacyBox that is relaying packets, the worst case regarding packet sizes is to be the first hop in a tunnel. In this case, the privacy box has to process the encrypted packets for all following PrivacyBoxes. One 160 byte packet is sent 50 times per second for VoIP communication (cf. [35]). These 160 bytes would be received as 618 bytes by the relaying privacy box in the six-hop testcase and sent to the next hop with a size of 554 bytes. This leads to a constant downstream of 30.9 kB/s and a constant upstream of 27.7 kB/s for one connection. Figure 7.8 shows the calculated throughput for up to 50 relayed TCP connections.

This can be considered an upper bound because of the amount of hops (six) and the amount of connections (50). According to Akamai's *State of the Internet report* [36] from the first quarter of 2015, the global average connection speed currently is 625 kilebyte per second. This would allow around 20 simultaneous connections. In order to allow users in countries with less well developed internet infrastructure to take part in the privacy network, the development of an additional module, allowing to impose limitations on the amount of relayed traffic, would be desirable.

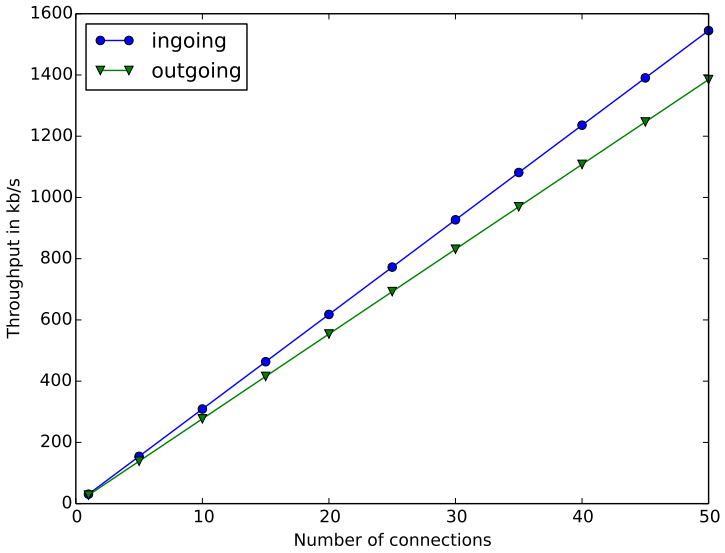

Figure 7.8: Overhead

7.2 Security and privacy

Beyond the performance related properties evaluated in the previous section, the extent to which privacy providing properties are ensured by the proposed system needs to be examined. This amounts to the question how the system fares when faced with both active and passive attackers.

7.2.1 Passive attacks

Passive attackers try to gain information without interfering with the traffic. They can be divided into those that are observing only one participant and those that are observing the whole network.

7.2.1.1 Passive local attacker

A passive attacker, observing one participant in the privacy network, can determine that the participant *is using the privacy network*. This is due to the fact that one specific port is used for privacy network traffic.

As far as the current design goes and to the extent of what is implemented in the prototype, a passive attacker might be able to *deduce if the participant is initiating a tunnel establishment* or if the participant is *just acting as a relay*. This depends on the

fact whether there are already multiple connections going through the PrivacyBox or not. If there are no connections, then an incoming connection followed shortly by an outgoing connection can be attributed to the participant acting as a relay. On the other hand, an outgoing connection without prior incoming connection can be understood as a connection establishment initiated by the participant.

This possibility will be removed by an additional module that was not part of this thesis: the cover traffic module. If every PrivacyBox opens and closes random tunnels through the Privacy network regularly, a passive observer can not as easily draw conclusions. On the one hand every PrivacyBox now always has multiple concurrent connections. Connecting incoming to outgoing streams as well as identifying initiated tunnels is thus made considerably harder. On the other hand, an observer now doesn't even know if a connection is used to transfer meaningful data.

In the current state of the PrivacyBox-Framework, this passive attacker might be able to *deduce the position in a tunnel* that a PrivacyBox holds. This is the case if the attacker is aware of the type of content (and thus possibly packet sizes) that is sent through the tunnel. In this case, the size of the encrypted payload might reveal information about the PrivacyBoxe's position in the tunnel (cf. section 7.1.2). In order to avoid this, a padding scheme needs to be introduced. This has the drawback of every PrivacyBox always processing the largest possible packet size. It is necessarry in order to avoid this kind of conclusion by a passive attacker nonetheless.

Another item that might be problematic are the DHT requests. If all requests are done by the initiating PrivacyBox, an attacker observing this PrivacyBox might be able to gain information from it. Even though these packets are encrypted, an attacker can see that a PrivacyBox is making the request and therefore deduce that it is the initiating PrivacyBox. This problem could be made less severe by caching IP addresses for a reasonable amount of time. In addition, implementing DHT-cover traffic could help. If both initiating and relaying PrivacyBoxes always send DHT requests when establishing a connection to another PrivacyBox, an observer can not know whether the information is used for the tunnel establishment (in the case of the initiating PrivacyBox) or simply discarded (in the case of relaying PrivacyBoxes).

This passive attacker can not see any content and can also only know about the relay nodes directly connected to the observed participant's PrivacyBox.

7.2.1.2 Passive global attacker

The next attacker to be considered is the much more powerful passive attacker observing the whole network. In addition to the previously mentioned abilities, this passive attacker might be able to *link initiator and target* of a tunnel. This is the case if the additional actions described in the previous section are not taken. The attacker can use

the techniques already described for every hop that the traffic goes through and thus draw comprehensive conclusions. If using cover traffic and a padding scheme, linking initiator and target becomes considerably harder. As every participant is simultaneously a user of and a relay node in the privacy network, even timing correlation attacks are prevented when using the proposed enhancements. A timing attack might try to connect a packet entering a privacy network to a packet exiting it by looking at correlations on both ends. In contrast to, for example, the Tor [4] network, there is no traffic entering or exiting the proposed privacy network. As a result, there is no way to correlate traffic entering and exiting the PrivacyNetwork.

An additional problem, when faced with a passive attacker observing the target host of a tunnelled connection, stems from the use of certificates. While there is no authentication of the relaying PrivacyBoxes, the end-to-end-connection needs to be authenticated. In order to provide this, the certificates used for the TLS/DTLS connection establishment need to contain information identifying both parties. If transmitted in the clear, a passive attacker might be able to *identify both initiator and target*. In order to prevent a passive attacker from accessing this information, an unauthenticated tunnel could be established to the target host first. Once this is done, another authenticated tunnel could be established inside the unauthenticated one.

7.2.2 Active attacks

An active attacker might try to compromise the privacy network by *running multiple malicious privacy boxes*. In doing this, an active attacker could increase the probability of a tunnel being built solely through PrivacyBoxes operated by the attacker. This would compromise the privacy of the users using these tunnels, as the attacker now knows who is communicating with whom. Depending on the size of the privacy network (i.e. number of participants), this approach might be costly. The more legitimate PrivacyBoxes are in the network, the less likely it is that a tunnel is built solely through compromised PrivacyBoxes. Due to the fact that an active attacker mostly targets specific users, this approach is even less promising. In order for the attack to be successful, not only any tunnel, but tunnels initiated by specific users would need to be built solely through malicious PrivacyBoxes.

As an extension of this attack, the active attacker might *manipulate the tunnel establishment*. A malicious PrivacyBox could build a tunnel through other malicious PrivacyBoxes ignoring the received connection requests. As the intermediate hops need to be anonymous, there is no way for the initiating PrivacyBox to check if the tunnel was really built through the PrivacyBoxes running at the requested IP addresses. The attacker could still not force a PrivacyBox to create a tunnel through a malicious PrivacyBox. If a connection is made through a malicious PrivacyBox on the other hand, the attacker could redirect the tunnel through other PrivacyBoxes controlled by him or her. The at-

tacker doesn't know at which step of the tunnel establishment he or she was introduced. As a result, he or she would need to guess at which point the connection to the final target PrivacyBox is requested and then stop to tamper with the tunnel establishment and connect to the real target. Otherwise, the initiating PrivacyBox realizes that the connection was tampered with, as PrivacyBoxes do authenticate each other end-to-end.

To sum it up: An attacker might be able to redirect a tunnel through attacker-controlled nodes. Doing this, the attacker has a high risk of being exposed.

In order to make this kind of attack even more impractical, a reputation system could be introduced. If a PrivacyBox reports PrivacyBoxes complicit in a compromised tunnel establishment, malicious PrivacyBoxes could be exposed. A problem with this approach is that all PrivacyBoxes need to be anonymous. This would make it impossible to globally connect a rating to a PrivacyBox.

Another kind of attack an active attacker could try is a *denial of service* (DoS) attack. This means that the attacker sends a lot of tunnel establishment requests to a certain PrivacyBox. In fact, the attacker tries to send as many requests as it takes to overwhelm the PrivacyBox so that it can't act as a PrivacyBox for other network participants anymore. Depending on the type of device the PrivacyBox-Framework is running on, this attack might be more or less challenging (cf. section 7.1.1.2). Currently, there is no system employed to prevent these kinds of attack. Possible solutions would be to limit the amount of tunneling requests a certain PrivacyBox can send to one PrivacyBox. Additionally, this behaviour could be included in the rating of the previously proposed reputation system.

The active attacker can also *circumvent the solution to the certificate information problem* introduced for the passive attacker. This is due to the fact that an unauthenticated TLS tunnel establishment can be attacked with a man-in-the-middle attack. For this to work, the attacker needs to be able to intercept the traffic. The attacker pretends to be the target PrivacyBox when communicating with the establishing PrivacyBox and vice versa. The attacker can then decrypt, read, encrypt and send onwards every piece of data.

In order to provide a way to at least detect a man-in-the-middle, the following approach can be taken: After establishing the unauthenticated secure connection and successfully establishing the authenticated connection through it, both parties could sign the keys that were used in the unauthenticated connection and exchange the signatures. This would allow them to verify that they used the correct keys during the unauthenticated part.

Another way to even avoid this problem might be found in using another channel to provide authentifying information about the participants.

One option would be to make use of the DHT. It is already the case that a PrivacyBox

providing a service has to store information for every contact that is supposed to be able to access the service (cf. [7]) in the DHT. How exactly the DHT could be used is beyond the scope of this thesis.

7.3 Summary

As mentioned at various points during this evaluation, there is still some work left to do for the proposed system to meet its goals. This is simply due to the fact that these missing features would have gone beyond the scope of this thesis. However, there is a clear path to solving most of these issues.

Apart from that, the evaluation has shown that the proposed system is able support TCP and UDP protocols, to achieve acceptable latencies while using multiple relay nodes and to hide as much information as possible from other relay nodes and attackers.

Revisiting Table 3.1 from chapter 3, Table 7.3 now also shows the entries for the PrivacyBox-Framework.

Service	**Type of traffic**	**Real-time**	**Third party privy**
Bitmessage	Only Bitmessage	No	No
Proxies	Limited	Yes	Yes
VPN	All	Yes	Yes
Tor	Only TCP	No	No
PrivacyBox	TCP & UDP	Yes	No

Table 7.1: Overview for presented services – revisited

Chapter 8

Conclusion and Outlook

This thesis showed that there are various currently existing approaches to obtain privacy on the internet. While these approaches may offer solutions that work for their specific use cases, none of them provides the ability to have anonymous VoIP communications and to obtain privacy for all types of application layer traffic leaving a device.

In order to develop a privacy providing network underlay that offers these features, a thorough analysis of what exactly could be expected of such a service was undertaken. In the course of this analysis, various attacks were presented that the system needs to be able to deal with. Building upon that, the requirements were worked out.

Following that, the main part of the work done for this thesis was designing a framework to fulfill those requirements and implementing a prototype based on that design. The framework makes use of various existing technologies in order to achieve the set goals. It uses TLS/DTLS for encryption, makes use of the libnetfilter_queue-module in order to intercept traffic and can be used in collaboration with OpenVPN in order to provide the possibility of running on a remote device.

The key element of the design is how it provides privacy to user's traffic: When a user wants to connect to another user in the privacy network, the system builds a layered tunnel to that other user. This includes establishing a secure connection to a random first relay node, then establishing another secure connection to a random second relay node, but through the previously established connection to the first relay node. This can be repeated multiple times until finally reaching the target user. Traffic sent through this tunnel will be encrypted multiple times and sent through all relay nodes participating in the tunnel.

The design of the system is divided into several modules, each accomplishing one particular task. The intercept-module acts as the root of the system. It enters the iptables-rules into the firewall and intercepts all packets after that. It does not do anything with the packets, but passes them along to the inspect-module. The task of

the inspect module is to be aware of all application-layer-modules and pass the packets along to them. Application layer modules finally decide about how to react to packets that match their protocol. This mostly means using the traffic relaying modules in order to build a tunnel to the target and sending all following traffic that is part of this connection through that tunnel. Another part that is mostly independent of the rest of the system is the server-module. It is responsible for relaying traffic that is received from other participants in the privacy network. This module decides whether the traffic is supposed to be relayed further, or whether it is targeted at a service residing in the local network.

The implementation based on this design was then evaluated. The evaluation found that - apart from issues with the implementation of DTLS tunnels - the design is able to satisfy the challenge. Latencies were largely within the scope of an acceptable range and a basic foundation of privacy preserving features are provided.

Due to the fact of the limited scope of this thesis, there is still work left to be done. In order to thoroughly fulfill the requirements regarding privacy, additional modules are needed. This includes modules providing cover traffic for both privacy traffic and DHT packets and the introduction of a rating scheme. Another problem that remains to be solved is providing a more efficient implementation of DTLS tunnels. This would either mean contributing this feature to the twisted open source network library or implementing a network library that natively supports DTLS. There is currently no python networking framework supporting DTLS that could be used instead.

Evaluating the system in a real world setting would also be beneficial and may provide further insight into possible improvements.

Bibliography

[1] M. B. Kaufman. (2013) A Guide to What We Now Know About the NSA's Dragnet Searches of Your Communications. [Online]. Available: https://www.aclu.org/blog/guide-what-we-now-know-about-nsas-dragnet-searches-your-communications

[2] Electronic Frontier Foundation. (2014) Government Documents and Admissions about Domestic Internet Backbone Surveillance. [Online]. Available: https://www.eff.org/pages/government-documents-and-admissions-about-domestic-internet-backbone-surveillance

[3] Deutscher Bundestag. (2014) 1. Untersuchungsausschuss (NSA). [Online]. Available: http://www.bundestag.de/bundestag/ausschuesse18/ua/1untersuchungsausschuss

[4] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[5] U. Mansmann. (2013) Telekom beginnt mit Umstellung herkömmlicher Telefonanschlüsse auf VoIP. [Online]. Available: http://www.heise.de/newsticker/meldung/Telekom-beginnt-mit-Umstellung-herkoemmlicher-Telefonanschluesse-auf-VoIP-1807580.html

[6] B. Schöntag, "Integrating Voice Over IP Into a Privacy Friendly Network," Bachelor Thesis, Technische Universität München, 2015.

[7] A. Kammerloher, "Address Resolution and Authentication for a Distributed Communication Underlay," Bachelor Thesis, Technische Universität München, 2015.

[8] J. Postel, "Internet Protocol," RFC 791 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1349, 2474, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[9] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460 (Draft Standard), Internet Engineering Task Force, Dec. 1998, updated by

RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. [Online]. Available: http://www.ietf.org/rfc/rfc2460.txt

[10] C. Eckert, *IT-Sicherheit: Konzepte - Verfahren - Protokolle*, 9th ed. De Gruyter, Oct. 2014.

[11] Statista Inc. (2011) Wie wichtig ist für Sie der persönliche Schutz Ihrer Privatsphäre im Internet? [Online]. Available: http://de.statista.com/statistik/daten/studie/205381/umfrage/stellenwert-des-schutzes-der-privatsphaere-im-internet/

[12] Pew Research Center. (2015) American's Attitudes About Privacy, Security and Surveillance. [Online]. Available: http://www.pewinternet.org/2015/05/20/americans-attitudes-about-privacy-security-and-surveillance/

[13] NSA. (2013) 3 XKEYSCORE slides. [Online]. Available: https://www.documentcloud.org/documents/894406-nsa-slides-xkeyscore.html

[14] P. Yee, "Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 6818 (Proposed Standard), Internet Engineering Task Force, Jan. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6818.txt

[15] The OpenSSL Project. (2014) OpenSSL. [Online]. Available: https://www.openssl.org/

[16] H. Kinkelin, *Autonomous and Robust Components for Security in Network Domains*, ser. Network architectures and services. TUM, Lehrstuhl für Netzarchitekturen und Netzdienste, Oct. 2013.

[17] OpenVPN Technologies, Inc. (2014) OpenVPN HOWTO. [Online]. Available: https://openvpn.net/index.php/open-source/documentation/howto.html

[18] J. Yonan. (2003) The User-Space VPN and OpenVPN. [Online]. Available: https://openvpn.net/papers/BLUG-talk/index.html

[19] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878, 6176, 7465, 7507. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[20] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347 (Proposed Standard), Internet Engineering Task Force, Jan. 2012, updated by RFC 7507. [Online]. Available: http://www.ietf.org/rfc/rfc6347.txt

[21] Netscape Communications Corporation. (1997) The SSL Protocol. [Online]. Available: https://web.archive.org/web/19970614020952/http://home.netscape.com/newsref/std/SSL.html

[22] N. Modadugu and E. Rescorla, "The Design and Implementation of Datagram TLS," in *Proceedings of ISOC NDSS 2004*, Feb. 2004.

[23] R. Brown. (2014) Dtls 1.0.1. [Online]. Available: https://pypi.python.org/pypi/Dtls

[24] H. Welte. (2014) The netfilter.org libnetfilter_queue project. [Online]. Available: http://netfilter.org/projects/libnetfilter_queue/

[25] J.-P. Lang. (2012) nfqueue-bindings. [Online]. Available: https://www.wzdftpd.net/redmine/projects/nfqueue-bindings

[26] Twisted Matrix Labs. (2015) What is Twisted? [Online]. Available: https://twistedmatrix.com/trac/

[27] D. C. Schmidt, "Pattern languages of program design," J. O. Coplien and D. C. Schmidt, Eds. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995, ch. Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching, pp. 529–545.

[28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[29] J. Warren. (2012, Nov.) Bitmessage: A Peer-to-peer Message Authentication and Delivery System. [Online]. Available: https://bitmessage.org/bitmessage.pdf

[30] C. Nath, *Key Issues for the 2015 Parliament*. House of Commons Library, 2015, ch. The 'Darknet', http://www.parliament.uk/business/publications/research/key-issues-parliament-2015/.

[31] Privax Ltd. (2015) How VPN works. [Online]. Available: https://www.hidemyass.com/how-vpn-works

[32] *ITU-T Recommendation G.114, "One way transmission time*. Springer-Verlag, 1996.

[33] Raspberry Pi Foundation. (2012) Raspberry Pi 1 Model B. [Online]. Available: https://www.raspberrypi.org/products/model-b/

[34] Verizon. (2015) IP Latency Statistics. [Online]. Available: http://www.verizonenterprise.com/about/network/latency/

[35] A. Cislak, "Analysis of modeling anonymized voice over ip traffic," Guided Research Project, Technische Universität München, 2015.

[36] Akamai Technologies. (2015) The State of the Internet / Q1 2015. [Online]. Available: http://spanish.akamai.com/enes/dl/soti/q1-2015-soti-fullreport-a4.pdf