

# Analysis of the I2P Network

## Information Gathering and Attack Evaluations

**Bachelor Thesis**

Degree course: [Computer Science]

Authors: [Jens Müller]

Tutor: [Prof. Dr. Emmanuel Benoist]

Experts: [Daniel Voisard, BAKOM]

Date: 16.06.2016

## Versions

Version	Date	Status	Remarks
1.0	16.06.2016	Final	Final version

# Management Summary

This thesis consists of three main parts, one giving an introduction to I2P, the Invisible Internet Project, one that presents I2P Observer, a software to collect information about I2P and one that evaluates different attack possibilities on I2P.

The I2P Network aims to provide anonymous communication between participating systems inside an overlay network, separated from other Internet traffic. It is written in Java with versions for Windows, Linux and Mac, providing HTTP- and HTTPS-Proxies for browsing and APIs to adapt applications to communicate over I2P. Multiple layers of strong cryptography are used to protect the content of packets and a mechanism called Garlic Routing, where multiple messages for the same destination can be encapsulated, to hide meta data. Its NetDB, a database distributed across participating peers - so called floodfill routers - contains all information needed to contact other users and services.

I2P Observer is a software written in Java to collect and publish data from the I2P network to provide historical data about it, with graphical representation of the most important facts to ease the detection of major changes. It extracts information from a locally running I2P instance including addresses, the number of floodfill routers, their known amount of entries in the NetDB and the software version. It then creates a website with the collected information, split into daily overview pages with accurate data and monthly ones where the average for each day is computed. I2P Observer is also easily configurable to collect more and more accurate data. It can be accessed via its website[11].

The main motivation for this thesis was an evaluation of I2Ps security, which was researched with 4 different approaches:

- Impersonation of a service: Can a user be redirected to a malicious server without noticing it? A Clickjacking attack which allows this under certain circumstances was found.
- Identifying I2P traffic: Is it possible to find I2P packets inside a network stream?
- Is Garlic Routing working: Can individual packets be traced through the network if there is little traffic?
- Recap a High Traffic Attack inside in a test setup: Send a high amount of packets towards a target and use the possibility to monitor the whole network to identify all participating peers and the destination.

The evaluation showed that I2P has many mechanisms built-in to strengthen its security. Its traffic is encrypted and hard to detect, Garlic Routing hides individual packets very effectively and although some attacks might be possible, they depend on many constraints.



# Contents

<b>Management Summary</b>	<b>i</b>
<b>1. Introduction</b>	<b>1</b>
1.1. Related Work . . . . .	1
<b>2. Introduction to I2P</b>	<b>3</b>
2.1. I2P Basics . . . . .	3
2.2. Technical I2P Overview . . . . .	5
2.3. Attacks on I2P . . . . .	11
<b>3. I2P Observer</b>	<b>13</b>
3.1. Program Design . . . . .	13
3.2. Methods . . . . .	17
3.3. Results . . . . .	20
3.4. Discussion . . . . .	21
<b>4. Adaptation of TOR Browser for I2P</b>	<b>23</b>
4.1. Methods . . . . .	23
4.2. Results . . . . .	24
4.3. Discussion . . . . .	24
<b>5. Test Network</b>	<b>25</b>
5.1. Network Planning . . . . .	25
5.2. Methods . . . . .	27
5.3. Results . . . . .	28
5.4. Discussion . . . . .	29
<b>6. Attacks on I2P</b>	<b>31</b>
6.1. Methods . . . . .	31
6.2. Results . . . . .	36
6.3. Discussion . . . . .	37
<b>7. Conclusion / Results</b>	<b>41</b>
7.1. Future Work . . . . .	41
<b>Declaration of authorship</b>	<b>43</b>
<b>Glossay</b>	<b>45</b>
<b>Bibliography</b>	<b>47</b>
<b>List of figures</b>	<b>49</b>
<b>APPENDICES</b>	<b>51</b>
<b>A. Configurations</b>	<b>51</b>
A.1. SSH Configuration . . . . .	51
A.2. Shell script to control all relays . . . . .	51
A.3. iptables . . . . .	52

<b>B. Examples</b>	<b>53</b>
B.1. I2P Observer . . . . .	53
B.2. Attacks on I2P . . . . .	54

# 1. Introduction

Providing anonymous communication over the Internet is an important, yet difficult task. As the global mass surveillance steadily increases, so are the capabilities of state agencies and corporations who try to collect as much data about every user as possible. The requirements towards software that allows the user to circumvent this tracking are very high, yet it is indisputable that such software is needed for many purposes, such as protecting whistle-blowers or allowing people to evade censorship.

Among other projects, I2P, the Invisible Internet Project, aims to offer this anonymity. Contrary to the well-known and -research TOR project [25], the number of active users of I2P is manageable and analyses of its security rare. This fact led to the choice to dedicate this thesis to the I2P network with the goal of getting to understand it better and possibly help improving its security.

This thesis consists of three major parts. Due to the fact that this is the first research done on I2P at Bern University of Applied Science, the first part of the thesis (Chapter 2) consists of an overview over the software with descriptions of basic operations as well as more detailed and technical ones needed to understand the following chapters.

Afterwards, a software to collect and publish statistical data about I2P is introduced with I2P Observer in Chapter 3. As analyzing the security of such a network and engineering possible attacks against it requires data, it was a consequentially decision to develop a program which gathers information about the size of the global I2P network and publishes it on a website. This also provides continuous data, giving the possibility to detect major changes in it.

The Chapter 4 briefly describes how to use I2P with the TOR browser, which provides very privacy-friendly settings and therefor is a good tool to protect the users privacy while browsing inside the I2P network.

Before specifying the different approaches that were taken to evaluate the security of I2P and possible attacks against it along with the conclusions drawn in the last chapter, which also is the third major part of this thesis, the planning and set-up of the test network is described in Chapter 5.

## 1.1. Related Work

For every part of this thesis, publishings on related work exist which were used to different extent and are presented here for each chapter.

The modes of operations of I2P are documented on its website [12] which was used as main source for the overview of the network.

Similar to I2P Observer, there was a project called bigbrother.i2p[2] which aimed at gathering information about I2P and publishing them inside the network itself. It allowed every user of I2P to provide data, making the results much more accurate than those of I2P Observer. However, the website of the project was no longer available during the work on this thesis and the site of the company behind it, Privacy Solutions [17], announced the relaunch of an improved version at some future date. The same company offers a software similar to TOR browser called the Abscond Browser Bundle which seems to be no longer available as its website [1] is unavailable and all discovered references to it indicate that the software is massively outdated, no longer considered secure and therefor should not be used anymore. Other projects of Privacy Solutions include an adaptation of I2P as a lightweight C++ client and an out-proxy service allowing to connect to the Internet from inside the I2P network. Both of these were not considered further for this thesis.

Successful attacks on I2P, which also give further understanding of its mode of operations, are published in multiple papers. The most relevant for this thesis are mentioned here. Herrmann et al. [23] found that the way I2P selects peers could be exploited to deanonymize an user. Timpanaro et al. [24] performed a monitoring study to estimate the size of the I2P network and for what purpose it is used. Egger et al. [21] showed an attack on the implementation

of the NetDB that allows to impersonate services and deanonymize a peer. Erdin et al. [22] discusses fundamental attack possibilities on anonymity-providing services such as TOR or I2P.



## 2. Introduction to I2P

### 2.1. I2P Basics

I2P is an abbreviation which stands for "Invisible Internet Project", a software to provide anonymous usage by hiding Meta Data.

#### 2.1.1. Concept of I2P

The goal of the I2P project is to provide the ability of privacy friendly communication inside an overlay network over the Internet, which means that no useful meta data can be collected from a packet and thus both the sender as well as its recipient stay anonymous. This goal should be achieved by creating a large network of so called routers which are nodes between which packets (so called messages) can be exchanged. To provide anonymity to all participants, a message is passed between multiple routers, making it very difficult for an observer who analyses data traffic to find the true sender and recipient as only the address of the last and the next router is visible. Because the content of the packet, which may include further routers and the recipient to forward the packet to is sent encrypted, it can not be analyzed and thus does not leak any information. Furthermore, an observer can not segregate between messages from the sender to the first router, between two routers or between the final router and the recipient. The concept of I2P is quite similar to the one of TOR Project [25], but differs in the basic goal of the project itself. While TOR aims to allow an anonymous access to the Internet to protect the users privacy and allows them to access blocked or restricted websites, making it some kind of an anonymity proxy, I2P is designed to protect messages inside its network, so all participants must have access to it. Connections to other networks such as the Internet or TOR are not intended, even though proxy services with that purpose exist.

#### 2.1.2. Installation

I2P is available for Windows, Mac and Unix-like operating systems. Guides on how to install it can be found on the I2P website [12]. As Debian Linux was used during this project, a short summary of how to install I2P on Debian will follow as it was done during the setup of the test network.

A packaged version of I2P is available via an additional repository, which needs to be added to the source.list of the packet management tool apt (or a separate list in /etc/apt/sources.list.d/ which is included by apt) as described in the I2P Installation Guide [13]. Afterwards, the public key that is used to sign the packages needs to be downloaded and imported to the apt-keyring via

```
wget https://geti2p.net/_static/i2p-debian-repo.key.asc
apt-key add i2p-debian-repo.key.asc
```

Then, after updating the sources with

```
apt update
```

I2P can be installed by the command

```
apt install i2p
```

When I2P was installed, it stores information such as the configuration or entries from the NetDB in the folder .i2p, which is located in the home directory of the user that starts the router software.

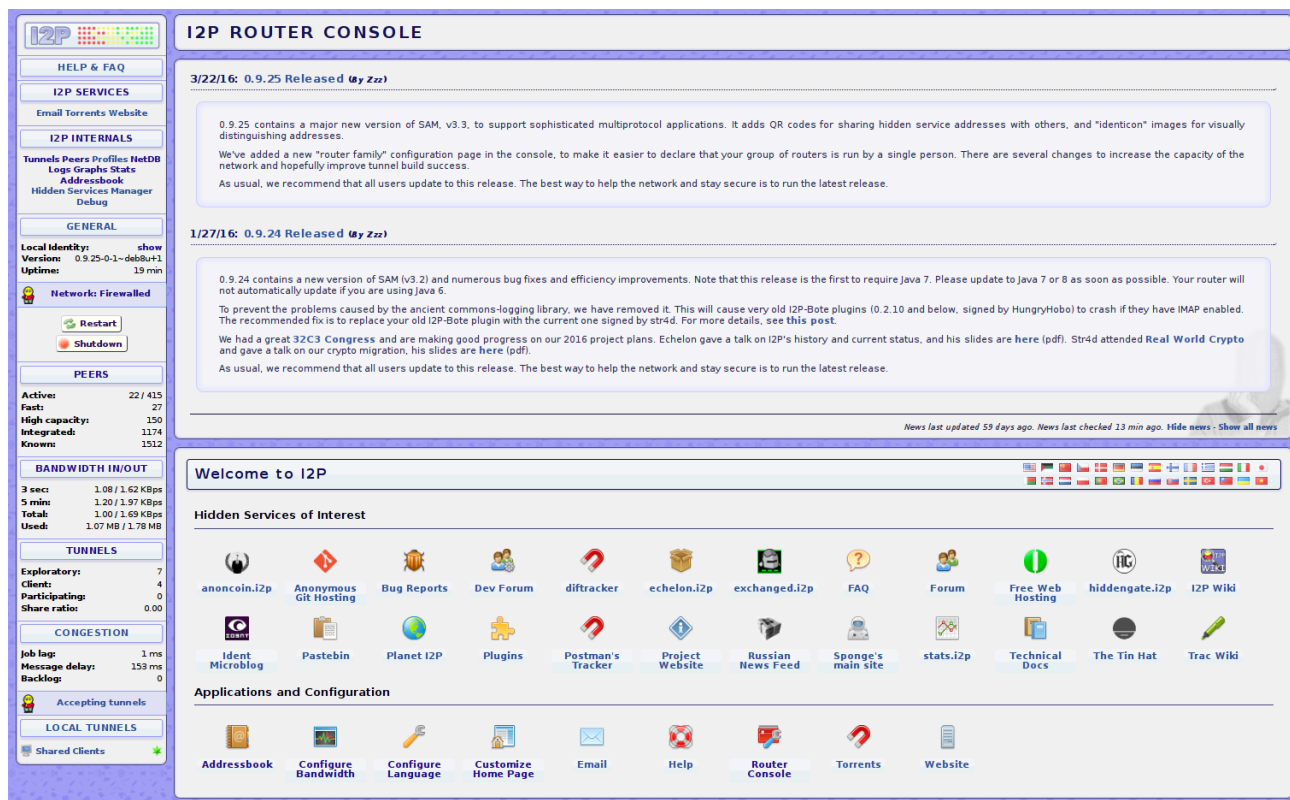


Figure 2.1.: Screenshot of the I2P Router Console

### 2.1.3. Starting I2P

Once installed, I2P can be started by the command

```
i2prouter start
```

This starts the I2P router, which begins detecting the network settings and Internet access, querying the built-in reseed servers for routerInfos and then starts building tunnels.

Information about the status of I2P can be viewed at the I2P Router Console (see **Figure 2.1**) and its configuration can be edited there. This is a website that is hosted on an internal web server and can be accessed at the address

```
http://localhost:7657
```

A screen-shot of it is displayed in **Figure 2.1**.

### 2.1.4. Available Services

I2P offers a lot of services already built into the software. Those include a web server, SMTP for anonymous mail communication, an IRC service for chatting and a BitTorrent client called I2PSnark. Some of the services are running by default, others need to be enabled via the Router Console first.

Additionally, I2P offers APIs which can be used by any program to communicate over the I2P network.

As the website used for tests inside the internal I2P network did not have special requirements, the built-in web server of I2P was used which consists of a simple page with instructions on how to set up an individual website.

### 2.1.5. The I2P Data Folder

I2P stores its data into the I2P data folder located in the user directory. For Linux, it resides in `/home/<user>/i2p`.

The folder contains all configuration and data of I2P in a logical structure. The `.i2p` folder includes the following subdirectories:

- `addressbook/` : contains settings for the address book like subscriptions or a log file
- `eepsite/` : folder of the web server included into I2P. The website needs to be placed inside the subdirectory "docroot"
- `keyBackup/` : contains a backup of all encryption keys used by I2P
- `logs/` : contains log files
- `netDB/` : contains the entries of the NetDB, grouped by the letter `r` and the first character of the routers hash (if the hash starts with `A`, it is placed in the sub-folder `rA`)
- `peerProfiles/` : Detailed Information about peers found by I2P, including tunnel properties. Could be used to create very broad statistics.

## 2.2. Technical I2P Overview

In this technical overview, the focus is to provide basic knowledge needed to understand this report, so it will often not go into the full details of every aspect. For further information or much detailed explanation, a link to the technical documentation on the I2P website [12] is provided in each paragraph.

### 2.2.1. Tunnels and Tunnel Creation

An I2P Tunnel is a set of routers in specific order used to forward a message. It normally consists of a gateway (the first router of the tunnel), a participant (the router in the middle) and an endpoint (last router in line). There are outbound tunnels for sending and inbound tunnels for receiving messages which both behave quite similar except the sending client is the gateway of the outbound tunnel while the receiving one is the endpoint of the inbound tunnel. Each client creates and maintains multiple tunnels simultaneously, one or more per direction for each application that uses I2P, configurable based on the need of privacy, as well as additional ones used by the I2P protocol itself for network maintenance such as NetDB queries (described in paragraph 2.2.4). It is possible to use multiple tunnels in parallel to increase privacy protection and bandwidth as well as changing their length.

The creation of an outbound tunnel is illustrated in **Figure 2.2**. The router which wants to build a tunnel retrieves a list of `routerInfos` with possible peers from the NetDB (see Chapter 2.2.4), selects the desired amount of routers (usually 2, but more are possible) based on their capabilities and puts them in favored order. It then computes creation messages for each participant in the tunnel where itself acts as gateway. Those are then sent to the next router (the participant) which learns that a tunnel should be created and that it should forward the creation message to the router which then becomes the endpoint. After doing that, it waits for the acknowledgement from the endpoint and sends it to the tunnel creator. If all succeeds, the tunnel can then be used to send messages to other users.

The creation of an inbound tunnel uses the same process (see **Figure 2.3**), but as described above, the creator acts as endpoint and the last router in the process becomes the gateway, which also acts as target address for messages intended for the receiver. An unique Tunnel-ID is used so the gateway can relate incoming packages to the intended receiver and forward them. As last step, the creator publishes the address of the gateway in the NetDB along with the Tunnel-ID.

This setup ensures that the involved routers do not learn any sensible information about the messages. They basically just forward packages from one router to another based on delivery instructions sent with them or the tunnel they belong to, without knowing who the true sender and recipient is.

More information about the implementation of tunnels can be found in the I2P documentation [9].

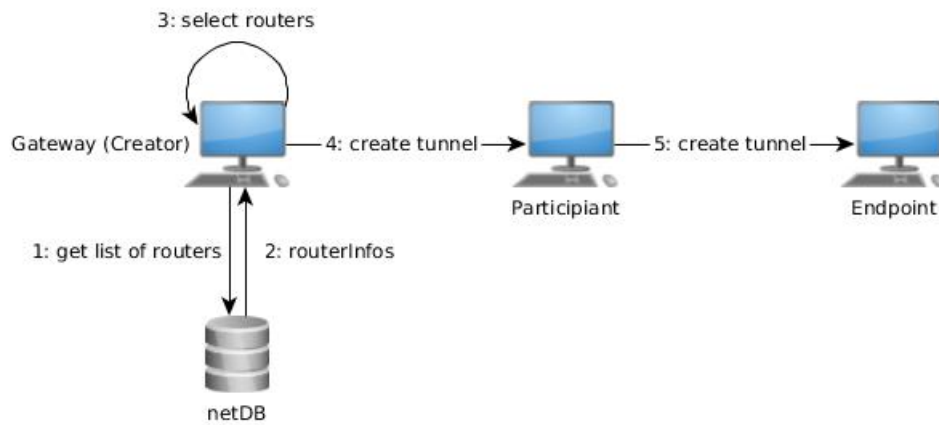


Figure 2.2.: Creation of an Outbound Tunnel

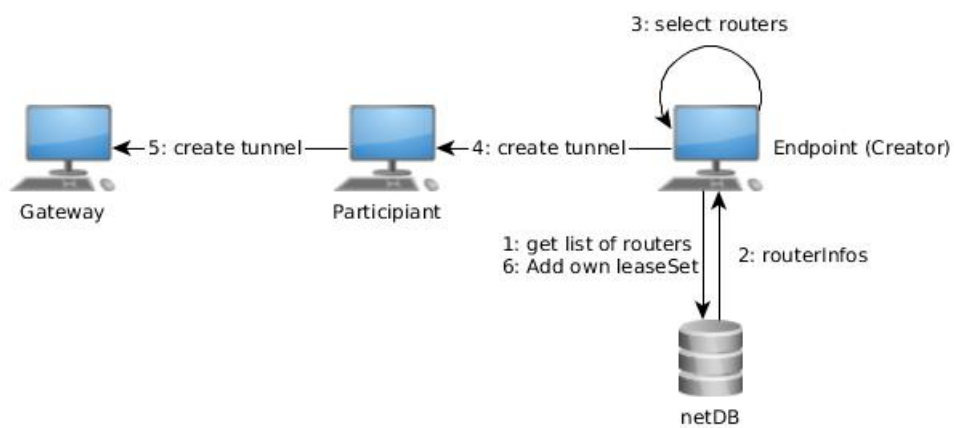


Figure 2.3.: Creation of an Inbound Tunnel

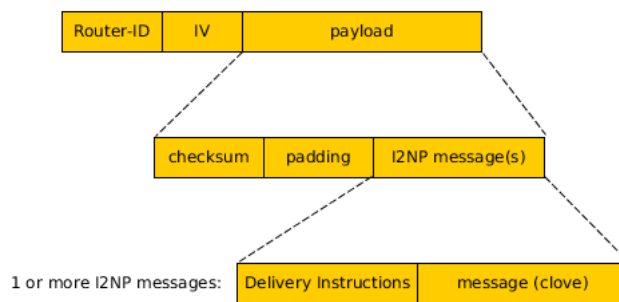


Figure 2.4.: Basic structure of a Tunnel Message

### 2.2.2. Message Structure

Information within the I2P network is exchanged in the form of I2NP (I2P Network Protocol) messages, mainly consisting of delivery instructions and payload, which can contain either a whole message or only parts of it due to size limitations.

To protect against Timing Attacks (explained in the glossary), multiple messages for a specific router are collected and combined into a so called Tunnel Message. This message consists of the ID of that router, the IV used for encrypting the payload and the payload itself which contains a checksum, padding (if needed) and the collected messages for the router in the format "Delivery Instructions" (where to forward this message) and the I2NPs itself (referred to as "Clove"). **Figure 2.4** shows a graphical representation of the Tunnel Message structure.

The complete specification of the Tunnel Message can be found in the associated I2P Documentation [10].

### 2.2.3. Message Encryption

I2P uses two different encryption schemes to protect information, AES256/CBC (symmetric encryption) inside the I2P Tunnels and ElGamal (asymmetric) for all other messages intended for a specific receiver. Every message is encrypted with the ElGamal public key of the receiver to protect its content, including management messages exchanged between routers.

Inside the tunnels, I2P makes use of a layered encryption so that each router only sees the information it needs, usually only the "Delivery Instructions" telling it where to forward the message to. The rest of the package, like further hops, the destination or the content of the message is encrypted with AES256/CBC. This layered encryption is done by the sender (outbound gateway) by first establishing a common, unique session key with each router of the outbound tunnel using ephemeral Diffie-Hellmann and then using that to encrypt the message step by step.

Additionally, messages that at some point would be exchanged in clear text are encrypted using Garlic Encryption (see Chapter 2.2.7) with ElGamal to prevent information leakage. For example, messages are not protected by layered encryption when they are routed between outbound endpoint and inbound gateway, so they are encrypted this way. Furthermore, establishing a session key between sender and recipient would create a link between them which is highly undesired.

The process of layered encryption for a single message is illustrated in **Figure 2.5**.

### 2.2.4. NetDB

The I2P NetDB is a database that contains information about all participants in the I2P network. It manages two sets of information needed to operate the network:

- **routerInfo:** Contains the information about a router, like its Identity (the ElGamal key, the signing key and certificate) or how to contact it (it's public IP address and port). This information is sent from the router to the NetDB as soon as it connects to the network and is identified by the SHA256 hash of the Identity.

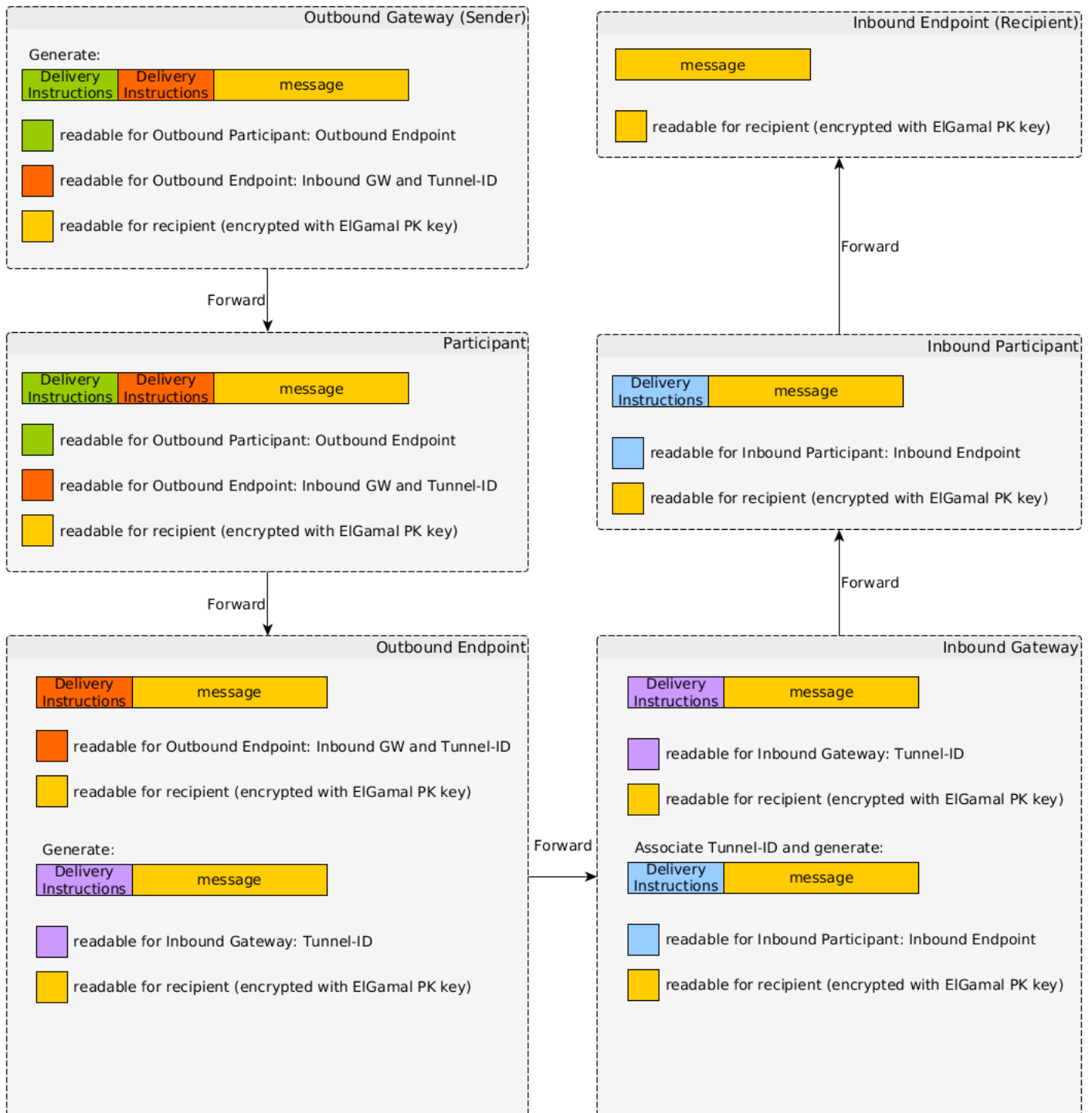


Figure 2.5.: Multi-layered encryption of a single message during the individual stages of transport as seen by every participant

- **leaseSet:** A leaseSet contains the information needed to contact a specific service, including a list of possible tunnel entry points (router IDs of the gateways and tunnel IDs of its inbound tunnels), the Identity (ElGamal key, signing key, certificate) and possibly additional encryption or signing keys. The leaseSet is also identified by its destination, the value of the SHA256 hash of the Identity.

The I2P software stores a local copy of the NetDB on disk in the folder NetDB/ inside the I2P data directory (under Linux: /home/<user>/.i2p), separated into sub-folders by the starting character of the hash of the routers key (so if the hash of the public key of the router is zLQOABCyMFbzgdbpZhXx5QBHablEVfmlxopLyKF3Q0Y=, it is stored in the file /.i2p/netDB/rz/routerInfo-zLQOABCyMFbzgdbpZhXx5QBHablEVfmlxopLyKF3Q0Y=.dat). When starting I2P for the first time, it contains a single entry, the routerInfo of the router itself.

The global NetDB is not hosted on a dedicated server but inside a decentralized database, shared among a set of routers (so called "Floodfill Routers") that are either selected randomly based on their bandwidth or volunteer for it. They collect information for the NetDB, synchronize with other floodfill routers and answer queries for routerInfo or leaseSet. This approach is known as "floodfill" in I2P. The NetDB uses a Kademlia DHT (distributed hash table) style approach where the identities of floodfill routers which in the address space are closest to the one of the destination are used to store the routerInfo or leaseSet. When a router wants to access an entry, it uses the same approach and asks the floodfill router whose Identity from his point of view is closest to the one of the wanted entry. If the queried server has that entry, it will provide the wanted information, otherwise it should know other routers closer to the target and refer the inquirer to them. This Kademlia-like approach is considered to be changed in the future, as practical attacks against it were found and published by Egger et al. [21].

Additionally, I2P uses bootstrapping to learn new routerInfo. This is known as "reseeding" and means that the router fetches a copy of NetDB entries from a source like publishings on other non-I2P networks (like on a regular website) or another router which already knows a large amount of information about the I2P network. Those sources are hard-coded into the router software and selected randomly. The copy consists of a zip archive which contains all known routerInfo files. This behavior is especially helpful after the start of the router as it has no knowledge about the network at this state and needs to learn information about other routers as fast as possible.

The reseeding can also be done manually by providing a zip archive with routerInfos. This method was used during this project to distribute the individual routerInfo files and introduce the peers of the internal I2P network to each other.

More specifications for the NetDB are described in the I2P documentations on NetDB [7].

## 2.2.5. Address Lookup

I2P uses a system similar to DNS called "susidns", where a hostname is translated into its actual base32- or base64-encoded address, called "destination". The destination is the SHA256 hash of the services Identity (2048 bit ElGamal Key, signing key and certificate). For example, the official site of the I2P Project can be accessed with the address i2p-projekt.i2p, the actual destination which is used to contact the server is however:

```
8ZAW~KzGFMUEj0pdchy6GQOOZbuzbqpWtiApEj8LHy2~O~58XKxRrA43cA23a9oDpNZDqWhRWEtehSnX
5NoCwJcXWWdO1ksKEUim6cQLP~VpQyuZTIllqwSADwgoe6ikxZG0NGvy5Fijgx4EW9zg39nhUNKRejYN
HhOBZKIX38qYyXoB8XCVJybKg89aMMPsCT884F0CLBKbHeYhpYGmhE4YW~aV21c5pebivvxeJPWuTBAO
mYxAIlgJE3fFU~fucQn9YyGUfa8F3t~0Vco~9qVNSEWfgrdXOdKT6orr3sfssiKo3ybRWdTpxycZ6wB4q
HWgTSU5A~gOA3ACTCMZBsASN3W5cz6GRZCspQ0HNu~R~nJ8V06Mmw~iVYOU5IDvipmG6~dJky6XRxCed
czxMM1GWFoieQ8Ysfuxq~j8keEtaYmyUQme6TcviCEvQsxyVirr~dTC~F8aZ~y2AIG5IJz5KD02nO6TR
kl2fgjHhv9OZ9nsh~I2jxAzFP6Is1kyAAAA
```

The translation is done locally using the address book which contains three parts: public, master and private. The public one includes a list of sites that is regularly updated by the I2P software on all clients and therefor offers the possibility to publicly announce a service. Every client can publish entries so other users can find hosted services if desired. Furthermore, it is possible to subscribe to other synchronization services to retrieve more public addresses. The master and private address book contains personal entries and can be used to store addresses that are not supposed to be publicly known.

When an user wants to access an Eepsite, the I2P software checks the address books to find the corresponding destination in the order private, master and public part. The first entry found is used, so an entry in the private part of the address book can overwrite one in the public part. If there is no entry for the address, the connection

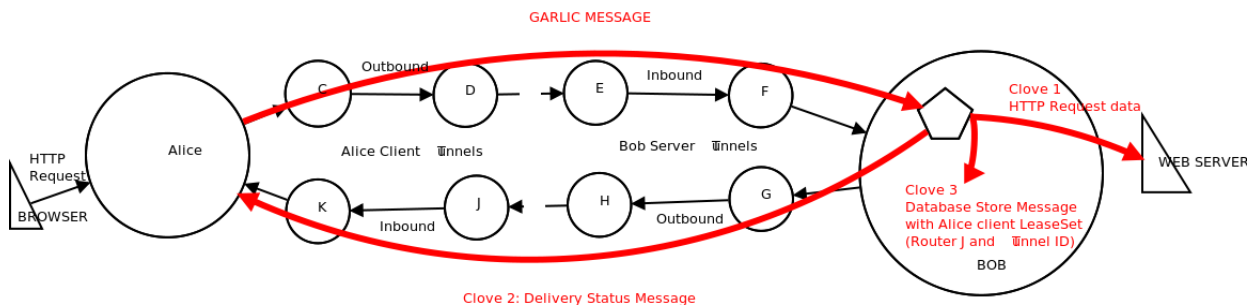


Figure 2.6.: Example for a Garlic Message between Alice and Bob.  
Source: <https://geti2p.net/en/docs/how/garlic-routing>

fails with an error page. However, some Eepsites offer so called Jump Services which can be queried to retrieve the destination they know for a hostname (if any). Four of them are included in the I2P software and the error page includes links to access them: i2host.i2p, stats.i2p, no.i2p and i2pjump.i2p.

## 2.2.6. Transport Protocols

To ensure a confidential, authentic and authenticated communication between routers, three different protocols have been developed for I2P over time providing this requirements. Two of them are currently used for delivering I2NP messages: SSU and NTCF.

**SSU** (Secure Semireliable UDP) is an entirely independently developed protocol based on UDP. It supports re-transmissions of unacknowledged packets (but only a fixed number of times, thus semi-reliable), as well as network specific tasks at startup like the determination of IP addresses, as well as detection and handling of NAT and firewall setups, like NAT traversal and UPnP.

**NTCF** stands for NIO (= new I/O) TCP and relies on Java TCP Transport for delivering packets, while connection related tasks like IP detection or NAT / firewall configuration are handled by SSU.

The full list of requirements for the transport protocols as well as the full specifications for SSU and NTCF can be found in the "Transport" section of the I2P Documentation [8].

## 2.2.7. Garlic Encryption or Garlic Routing

Garlic Encryption, also described as Garlic Routing or simply "Garlic", is a reference to the Onion Routing Project [16] which was the predecessor of the TOR Project and introduced the concept of routing messages through a tunnel using layered encryption. The exact meaning of "Garlic" is not specified by I2P, instead it is used to describe three different principals:

- **Layered Encryption:** As already mentioned in Chapter 2.2.3, messages are protected using multiple layers of encryption while they travel along a tunnel. This should prevent individual hops to learn more information about the message as needed to forward it.
- **Message Bundling:** In Chapter 2.2.2, a basic overview over the message structure and the idea to bundle multiple messages with the same target was introduced. Each individual message or part of it is referred to as "Clove" in I2P, while the whole bundle, also known as Tunnel Message, is called "Garlic Message".
- **ElGamal/AES-Encryption:** "Garlic Encryption" can simply describe the usage of ElGamal or AES Encryption for messages, like tunnel messages as described in Chapter 2.2.3 or NetDB information.

**Figure 2.6** shows an example of a typical Garlic Message between Alice and Bob. It contains 3 individual messages (Cloves): The request to a HTTP server, a delivery status message to acknowledge the reception of the request and a leaseSet of Alice so Bob knows where to send the answer to (as discussed in Chapter 2.2.4).

The I2P documentation [6] contains the full specifications and explanation of Garlic Routing.



## 2.3. Attacks on I2P

Due to heavy research, many attacks on the TOR network were found that use theoretical weaknesses (like Timing Attacks or Package Manipulation) or faulty implementation (like unwanted information leakage). A lot of them can probably also be applied to I2P as it uses similar principles as TOR. The article from Erdin et al. [22] offers an overview over possible attack vectors for both of these networks. As there was not as much research on I2P as on TOR to evaluate these attacks, the protection of it against these or additional, not yet discovered ones can not be seen as much proven as for the TOR network.

### 2.3.1. Attacks chosen for this Thesis

In this thesis, potential attacks were evaluated to analyze their threat to the I2P network. These attacks were:

- **Forgery of leaseSets or routerInfos:** Can a recipient be impersonated by creating a new or updating an existing leaseSet?
- **Identification of I2P traffic:** Can traffic of the I2P network be identified?
- How much traffic is needed that message bundling in Garlic Routing successfully hides meta data?
- **Usage of a High Traffic Attack** to identify the recipient by sending a large amount of data to him and observe the network to see where those messages are routed (visible by the sheer amount of packets), similar to a Timing Attack.



## 3. I2P Observer

To gather facts about the size and structure of the I2P network, the goal of I2P Observer was to extract information and publish it on a website. No further requirements were defined towards its structure, work flow or design, neither what data should be collected and which programming language should be used. The complete program and its processes had to be planned, designed, programmed and tested.

The latest version of I2P Observer can be found on its website[11].

### 3.1. Program Design

#### 3.1.1. Information to extract from I2P

There exists a variety of information available for extraction from I2P. A lot of it is already displayed in the I2P Router Console with the limitation that there is only the current state displayed, without any historic values. To provide continuous data which for example can be used to retrace major changes in the I2P network, I2P Observer should extract information from I2P frequently and publish it on a website.

From the many available information, the following were considered important and possibilities to gather them were evaluated:

- **Number of active routers:** From the available NetDB entries, the amount of currently active routers can be estimated.
- **Routers per country:** The addresses of the routers can be used to determine the country in which they are running and to create a statistic.
- **Number of Floodfill Routers:** The amount of routers that participate in the NetDB can be collected from the NetDB entries based on the published capabilities.
- **Number of public Eepsites:** The amount of entries in the public address book of I2P, which is synchronized with every I2P Router, can be collected by analyzing the .txt-file used for that. As it did not seem to experience many modifications, it was decided to not include this into the program.
- **Services used over I2P:** Although a statistic about services in the I2P network would be very useful, there is no possibility to extract such information from the NetDB or the address book (which only includes the public addresses from I2P itself). The only possibility to obtain information about services used inside I2P would be to collect leaseSets from the NetDB and then fingerprint the running services, for example with port scans, banner grabbing etc. This requires a lot of resources and time, would be very noisy (and therefore easily detectable) and could be seen as possible attack on the network. Due to this considerations, information about running services were not collected for I2P Observer.

#### 3.1.2. Concept of I2P Observer

I2P Observer consists of 3 main phases: Import, Conversion and Output creation. It creates a daily overview which lists all collected data for the current day. Additionally, a monthly overview is created by calculating the average values for every day and storing them in a monthly list.

## Import

The Observer uses the files in the NetDB folder of the I2P router, under Linux they are located in the folder /home/<user>/.i2p/netDb/. This makes it necessary that an instance of the I2P software is running to collect new information. The idea to gather data this way was inspired by an entry in the weblog TheHackerWay [19].

In a first step, the files are read into the Java program and imported as I2Ps own RouterInfo class. This allows for an easy extraction of all available information in the next step. This method also does not require any modification in the I2P software itself and therefor should be compatible with further versions, unless the fundamental concept of how NetDB data is handled is changed. If a new version of I2P is released, it can be provided to I2P Observer which should work fine without alteration.

If there are more than 20 errors during the import of routerInfo files or more severe ones, it is marked in the NetDbEntries class and later displayed on the website to indicate that the specific dataset contains more inaccurate data than others.

## Conversion

The Set of RouterInfo objects is then used to read the information from each one and store it in the custom data class RouterInfoDataset, together with the date or time of the reading. These datasets are then collected in the class RouterInfoStatistic where one object contains all RouterInfoDatasets for either one day or the daily average values for one month.

## Output Creation

The collected data in a RouterInfoStatistic is used to create a HTML page with two graphs and three tables, one for the NetDB data, one with a list of all versions found and one which lists the countries in which routers are operated, determined by the GeoIP Lite library [5].

For the daily overview, the format of the filename is YYYY-MM-DD.html, for the monthly one YYYY-MM.html.

### 3.1.3. Structure of Program

I2P Observer consists of multiple Java Class files. An overview of the classes can be seen in the Class Diagram (**Figure 3.1**), the work flow of the program is described in the Sequence Diagram (**Figure 3.2**).

A basic description of all classes is given below. I2P Observer uses five data classes to collect and or store data. These are:

**ExtractedRouterInformation:** This data structure is used by the RouterInfoAnalyzer class to store all information extracted from the NetDB routerInfos which is later used to create the statistic. This includes the hash of the router, its IP addresses, information on whether or not it runs as a Floodfill Router, and if it does, routerInfo and leaseSet it knows, as well as the version of I2P it is running.

**NetDbEntries:** Data structure that contains a list with all RouterInfo imported from the NetDB together with a boolean indicating if there were errors during their import, as described above.

**RouterInfoDataset:** Contains all data collected during one run of I2P Observer together with its date or time and a boolean to indicate import errors.

**RouterInfoStatistic:** A collection of all RouterInfoDatasets that belong to one page which is either a daily or a monthly statistic. It includes the date that it represents and a boolean to indicate whether it is a monthly or a daily one. It also allows to compute an average from a set of datasets.

**ObserverProperties:** Class that contains all settings for the I2P Observer like working directories, login credentials for the FTP upload, the title used on the output page or the template code for it. It also supplies the variable names for the entries in the RouterInfoStatistics to ensure a uniform and therefor persistent usage in all classes.

Additional to the data classes, the following ones are used:

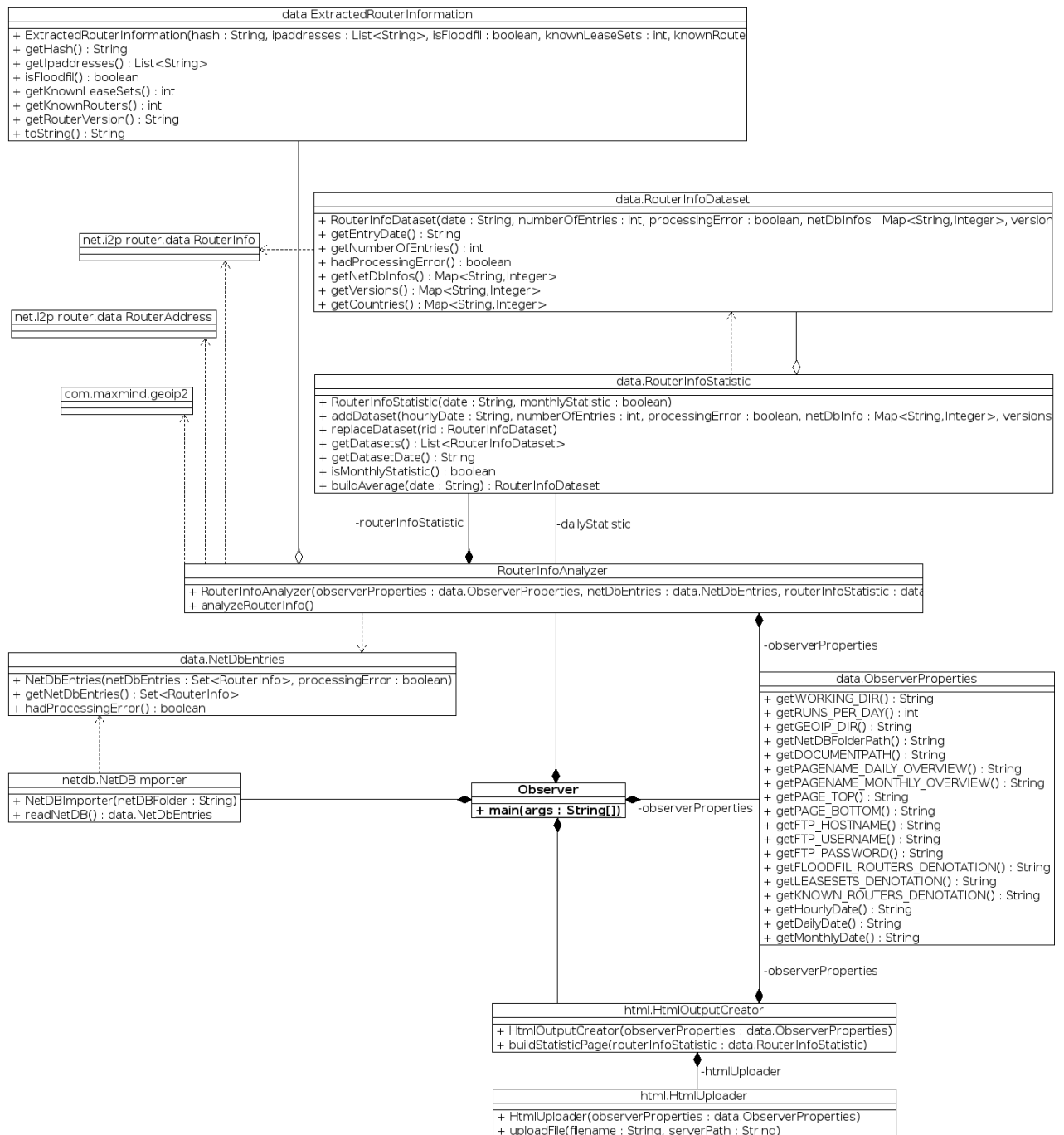


Figure 3.1.: Class Diagram of the I2P Observer

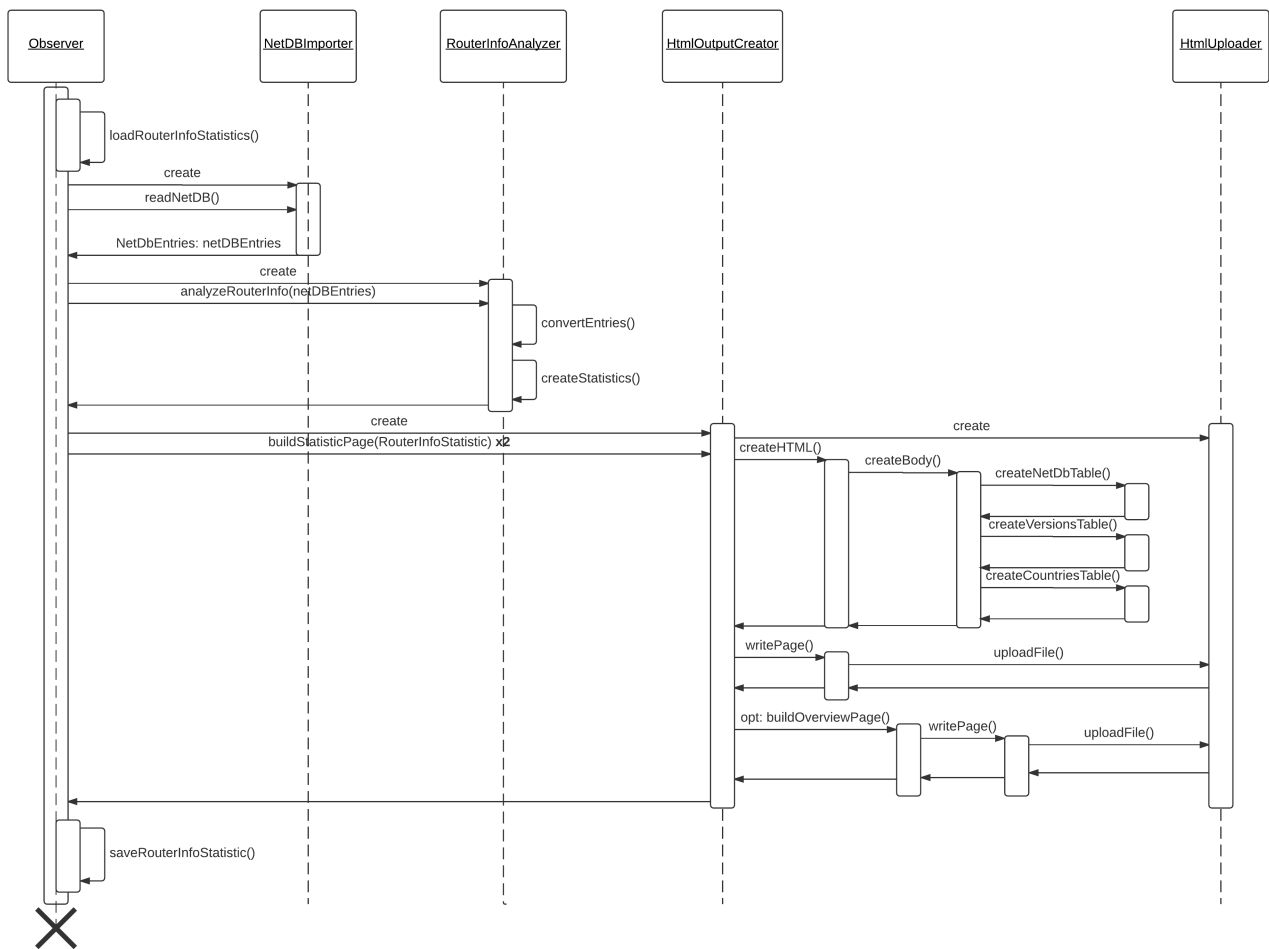


Figure 3.2.: Sequence Diagram of the I2P Observer

**Observer:** Base class of the I2P Observer. It imports the data of previous runs, handles the import from the NetDB folder and the creation of the current statistic together with its page both for the daily and monthly statistic and then saves the data back to disk.

**NetDBImporter:** This class reads all files in the I2P NetDB folder from disk and imports them into an object of NetDbEntries.

**RouterInfoAnalyzer:** The creation of the RouterInfoStatistics are done here. The class receives the Set of RouterInfos created by the NetDBImporter, converts them into ExtractedRouterInfos and creates a RouterInfoDataset by collecting all entries that belong together. During this step, it also initializes the GeoLite database to resolve the IP addresses to the appropriate country.

**HtmlOutputCreator:** Creates all HTML pages from the collected data and uploads them to the web server via FTP using the HtmlUploader. Based on the date, the overview and archive pages are created if needed together with the ones for the current statistics (both daily and monthly).

**HtmlUploader:** Used to upload the created HTML pages to the web space via FTP.

Figure 3.2 shows the work flow of I2P Observer. The main class Observer loads previous entries (serialized objects of RouterInfoStatistic), then imports the data for the current run by creating and using an object of NetDBImporter. This returns an object of NetDbEntries which contains a list of all imported RouterInfo as well as an indicator

(boolean) if there were larger errors during import, which is handed to the RouterInfoAnalyzer class to convert the entries, create a statistic in the form of a RouterInfoDataset and store it in the RouterInfoStatistic. Observer then creates and uses HtmlOutputCreator to create the daily statistic page and, in a second step after computing the average entry for the current day via RouterInfoStatistic, the monthly one. During the creation of the HTML page, HtmlOutputCreator also creates an updated overview site as well as the archive page if needed and then uploads all files via FTP to the web server using the HtmlUploader class. When HtmlOutputCreator has finished, Observer saves the current files to disk and exits.

### 3.1.4. Structure of Website

The website of I2P Observer is based on the Bootstrap Theme[3], with self-hosted jquery and Awesome font for privacy reasons. To create and display the charts on the statistics pages, Morris.js[15] and Raphaël JavaScript library[18] is used. Besides the pages created by I2P Observer, it also consists of a few pages created by hand, since they do not change and hold basic information. These are:

- **index.html**: The start page of the website
- **about.html**: A site with a short introduction about how I2P Observer works
- **help.html**: A Help page that explains the terms used on the statistic pages
- **impressum.html**: Contains contact information and mentioning of third party library used in the project

Apart from the files needed for the rendering of the website in the /themes/ folder, all other documents are located in the /statistics/ folder and are dynamically generated by I2P Observer. This includes the daily and monthly statistic pages, the overview pages for them and the archive pages.

The file structure on the web server looks as follows:

```
-/ : file root
|- theme/
|  |- bootstrap/ : contains files of Bootstrap theme
|  |- font-awesome/ : contains files of font Awesome
|  |- jquery/ : contains jquery files
|  |- morrisjs/
|     |- morris.min.js : Morris.js library
|     |- raphael-min.js : Raphael library
|- statistics/
|  |- YYYY-MM.html : statistic for one month (e.g. 2016-04.html for April 2016)
|  |- YYYY-MM-DD.html : statistic for one day (e.g. 2016-04-01.html)
|  |- archive_last_month.html : overview page of daily statistics for last month
|  |- archive_last_year.html : overview page of monthly statistics for last year
|  |- overview_daily.html : overview page of daily statistics for the current month
|  |- overview_monthly.html : overview page of monthly statistics for the current year
|- about.html
|- help.html
|- index.html
|- impressum.html
```

## 3.2. Methods

### 3.2.1. Information to extract from I2P

The first step to build I2P Observer was to create an overview about which data can be gathered and how to collect it, before evaluating how useful it can be. The NetDB folder of an I2P installation was soon found to be the major source of information, as it contains a lot of data about the network. The evaluation of collecting data from the I2P address book, which contains all official I2P Eepsites resulted in the decision to not use it as nearly all non-official services are not listed in it and therefore it does not offer interesting information. Additionally, the content of the

address book does not change very often, so including it in I2P Observer would have meant additional effort with nearly no useful result.

The analysis of the data in the NetDB folder showed that it contains at least the IP address of a router, its capabilities (abbreviated as "caps" in I2P), its version of I2P and, if it is running as floodfill router, the amount of leaseSets and routerInfos it knows. As all of this information is quite interesting and suitable to create a characteristic of the network, they were chosen to be included into the I2P Observer. The first approach was to collect the NetDB entries from the reseed peers used by I2P to gather a basic set of routerInfos right after the start, as it is described in Chapter 2.2.4. The list of those servers is build into I2P and can be edited in the reseeding settings of the I2P configuration. However, the attempt to obtain data from the servers mentioned there using tools like wget or curl with a simply GET request failed. Trying to figure out the correct query by analyzing the I2P source code (the query is done by the class `net.i2p.router.networkdb.reseed.Reseeder` which uses `net.i2p.util.EepGet` and `net.i2p.util.SslEepGet`) did also not succeed due to the very complex structure of the source code. A different approach to retrieve the correct queries by analyzing the network traffic of I2P with Wireshark failed as well, on the first try due to the TLS encrypted connection. After editing the entries and options so I2P only uses HTTP requests, it was possible to find the query ("`GET /i2pseeds.su3`"), but the replies from the servers to both I2P and own queries with wget and curl resulted in error codes as replies from the servers (mostly HTTP 403 and 404, but also more rare ones like HTTP 410 GONE). As this probably indicates that the reseed servers are not working as intended by I2P, this approach to retrieve NetDB entries was dismissed and a way to import the local files of an I2P installation was explored. Research on how to import NetDB entries into Java led to a website called TheHackerWay[19], where a similar task was performed for a program which uses the NetDB information for a different purpose. The concept explained there was chosen for I2P Observer as well, as it was both simple and independent of I2P in a way that no modification of the I2P software is necessary. The import method is described in Chapter 3.1.2.

Using I2Ps own RouterInfo class to extract the information provides all data available in the NetDB itself, so I2P Observer can be easily extended to include more statistics. Some possibilities for extensions are discussed in Chapter 3.4.1. Chapter 3.1.1 discusses which data was finally chosen for the I2P Observer from the information contained in the NetDB.

### 3.2.2. Java Program

I2P Observer was programmed in Java because I2P itself is written in the same language, which allows to use some of its classes when needed. As already mentioned previously, the first step to build I2P Observer was to create the class `NetDBImporter` which reads all entries in the NetDB folder and imports them into a list of `RouterInfo`.

Having the data imported from the NetDB as `RouterInfo`, the next step was to convert it. This was first done in a `NetDBConverter` class, which later was integrated in the class `RouterInfoAnalyzer`, as it was not needed elsewhere. To store the information, different data classes were created, like `ExtractedRouterInformation`. It holds the entries from the converted `RouterInfo` until they are processed by the `RouterInfoAnalyzer` class which collects all information from the imported NetDB by iterating over the list of `RouterInfo` and adds the data to a new list. For the creation of the country statistic, the third-party library `GeoLite2` [5] was included, which translates an IP address to the country it belongs to.

With a basic importing and converting tasks done, the base class `Observer` could be created and used for a test run which printed the result in the console. This showed that the basic concept worked fine, although some adjustments and corrections were needed in the source code.

The major work was to adjust the program so it efficiently creates HTML pages. The basic concept for creating them in the `HtmlOutputCreator` class was to create a String containing all the HTML code with a `StringBuilder` and then write it as file on disk. This proved to construct the pages as wanted and using formatting characters gives the HTML code some structure. The first version of the output page consisted of very simple HTML code which only showed three basic tables containing the collected information, one for general data from the NetDB, one counting the versions and one for an overview over all countries in which routers are operated. To be able to create the individual tables one after another, the structure of how `RouterInfoAnalyzer` stores its results was split up to separate them into one of three map structures representing the tables used on the web page. In each map, the name of the attribute (e.g. "leaseSets", "0.9.25" or "Switzerland") is used as key and depending on its meaning the corresponding value is used to either sum up all entries or to act as counter. This method makes the handling of the data very easy when creating the web page, especially as the usage of `TreeMap` automatically sorts the map. As the tables with the country statistics contained many entries, its orientation was flipped so the individual countries were listed in rows instead of columns. Although this made the creation of the table more complex, the



result looked more appealing, so the table with the version statistic was changed to that style as well.

In order to have an easy and fast way to adjust parameters like folders or specific settings, the class `ObserverProperties` was created which contains all options to configure I2P Observer. It is also used to provide uniformed variable names throughout the program, like the keys of the maps or date and time formats in the data classes.

The next step was to make the data persistent, so I2P Observer can be run when it should collect a new dataset and not having the need to run constantly. It then reads the previous collections from disk, imports the current data, creates the web page with the statistics and saves all data to disk. As no special requirements are needed for the persistence, this could easily be done by implementing the Java interface `java.io.Serializable` in the `RouterInfoDataset` class. To save daily and monthly statistics in a single object, the class `RouterInfoStatistic` was created which stores all related `RouterInfoDatasets` together with the corresponding date. It also contains a boolean to differentiate daily and monthly statistics, which in the beginning was used to load, handle and store the monthly statistic only once per day, but was later used for different purposes.

After adding a total entry to the tables, the decision was made to separate the daily statistic page by date, so it would not have to many entries. This meant creating a monthly overview page which links to the individual sites with the daily statistics. Furthermore the date which is used to create the links on this overview page needed to be added to the `RouterInfoStatistic` class as the individual `RouterInfoDatasets` only contain the exact point in time they belong to (which for daily statistics is only the time, but the date of that day is needed). When creating the overview page, the program checks if a website for each day exists first before adding it to the page to prevent dead links. Equivalent changes were made for the monthly statistics as well with a yearly overview page. To provide those with more accurate data, the method `buildAverage()` was created which computes the average value per day for all entries so a spike in one measurement would not falsify the statistic.

### 3.2.3. Website

Once the basics of I2P Observer were complete and it created simple tables with the data, the next step was to build a visually appealing website to represent the information. As base, the Bootstrap Theme [3] was used and implemented with a self-hosted jquery and the Awesome font. After creating some test pages by hand which included the HTML code of the tables from I2P Observer, the source code of the site was analyzed to find the fragments that can be wrapped around the output of the program to build a complete page. Those were then included in `ObserverProperties` as `PAGE_TOP` and `PAGE_BOTTOM` constants and are included when building the String with the HTML code of the web page (see previous chapter).

As the sites do not only need to be created but uploaded to the web server as well, the next step was to implement a possibility to upload files via FTP, done in the `HtmlUploader` class. To keep the dependency from other libraries as low as possible, the method described at CodeJava[4] was chosen which works well with the standard classes provided by Java.

To offer a fast overview over the collected data on the web page, a graphical representation was implemented using `morris.js` [15] to construct charts and the `Raphaël` library [18] to display it. I2P Observer adds the code for that to the site, creates an array with the data and embeds it at the end of the page. In a first step, a single chart with the entries of the `NetDB` was created as this table is considered to have the most important information. As the amount of `leastSets` (usually around 60000 - 80000 and more) is much higher than the other entries (the normal amount of routers is located around 5000 to 6000 and floodfill routers around 700 to 800), even rather big changes to the latter ones were difficult to see. To resolve that issue, a second chart was added which shows the values for the router entries only. The number of analyzed `NetDB` entries was later added to this chart to provide a sense about the ratio between those and the total amount of routers.

The decision to group the daily statistics by month and the monthly ones by year made it necessary to include an archive version for the month or year prior to the current one. Otherwise, all entries of the previous period could not be accessed anymore. This is done in I2P Observer by creating the overview page a second time with a different name when needed (which is at the first run of a month or a year). Although this could also be done by simply copying files, this would mean to implement this file handling either with an additional script which needs to be maintained or by adding extra code to I2P Observer, so this solution was considered more elegant.

### 3.2.4. Running I2P Observer on a Virtual Machine

As I2P Observer is intended to run on a host on which the I2P software is running as well, an additional virtual machine was created for it on the server which hosts the test network. On it, a Debian system together with I2P and Java 8 was installed as described in Appendix B.1.2 and different settings were tested for I2P to find the optimal configuration where the I2P software does not use too much bandwidth but is still able to learn new NetDB entries. This seems to be the case when it shares at least 12 KB/sec as bandwidth and does not run in hidden mode. Furthermore, the router needed to be forced to act as floodfill router in the advanced section of the configuration to collect new entries.

### 3.2.5. Evaluation of collected Data

To get a comparison to the data that the I2P Observer running on the VM collects and to evaluate if and if so which impact the fact that it runs behind a firewall has, a second host was set up and connected to a private Internet connection where port forwarding could be enabled on the firewall. This showed that the precision of I2P Observer depends on three factors: The accessibility of the router from the Internet, the shared bandwidth of the router and the performance of the system it runs on. All of these factors influence the ability of the router to learn about new peers during its participation in I2P tunnels. If the router is not reachable from the Internet due to a firewall, it is harder for other routers to contact it to build new tunnels. If it shares only a small amount of bandwidth, it can not participate in many tunnels and therefore only learns about the few peers of them. The same applies to the performance of the host system. Participating in tunnels needs a lot of cryptographic operations, so if I2P Observer runs on a system with little performance, only few tunnels can be participated in as well. To give some hard facts observed during the project: The original instance of I2P Observer that ran behind the firewall on average knew about 2000 to 2200 individual routers at a time, which was about 30 % of all routers. This was not influenced much by performance or shared bandwidth as increasing both of these factors only increased the number of peers it knew to around 2200 to 2500 or 35 - 40 %. The second instance of I2P Observer ran to evaluate the data as described in this chapter learned around 4000 peers (about 60 %) at start. This number increased to 4500 to 5000 and more after moving it to more powerful hardware as the performance of the CPU (an old Intel Atom processor) turned out to be the bottleneck, so it knew about approximately 70 % of the routers of the network at this time.

So the biggest factor that limits the capabilities of I2P Observer seems to be its (in)accessibility from the Internet due to firewalls.

Another observation is that I2P occasionally seems to lose a large amount of known peers and fails to reseed, so the amount of NetDB entries I2P Observer can analyze decreases drastically. Restarting I2P seems to solve this problem, so implementing a regular restart process, for example via cronjob, may prevent this situation.

As conclusion, I2P Observer should run on a host with an I2P instance that is directly accessible from the Internet via port forwarding, which provides enough bandwidth (250+ KB/sec Up- and Downstream are desirable) and performance to be most accurate. It should also support both IPv4 and IPv6 as both are used to communicate with other routers. Furthermore the I2P router should be restarted regularly to prevent unusual behavior.

## 3.3. Results

### 3.3.1. Java Program

The creation of I2P Observer was mostly straight forward and could be done without major issues. As described in the previous chapter, the biggest problem was to find a reliable source to extract the information from. Once the decision to use the NetDB folder of the I2P router was made, the program grew step by step without bigger issues and with clear structures of what part was to be implemented next. There are some additional features that could be added to I2P Observer discussed in Chapter 3.4.1 which could not be implemented due to time constraints. A refactoring of the source code would surely improve its readability and the performance of the program, but had to be skipped as well due to the lack of time.

Basic test classes were created for the parts of I2P Observer which are feasible to be tested, which in the end only includes the RouterInfoStatistic class with its buildAverage() method and the RouterInfoAnalyzer class. Testing

other data classes was not considered suitable as they only contain getter and setter methods. For NetDBImporter, a test for the part which imports the RouterInfo objects is difficult to implement as it means to operate with the quite complex structure of this I2P class and testing the simple part of importing a file from disk does not seem useful. Also, errors in this class would either result in an exception or in incorrect data which is noticeable on the statistic page, like invalid version numbers. Similar considerations apply to the HtmlOutputCreator and HtmlUploader classes. If the first one would not work properly, the resulting HTML code for a site would be mis-formatted which would be instantly noticeable when looking at the page. Errors in the HtmlUploader class would result in a page not being uploaded, which would also stand out. As the main class Observer coordinates the usage of the individual classes, it needs to rely on them for testing purposes which makes tests difficult.

The design decision to build I2P Observer as a one-shot program (which means that it runs once and then exits) offers some advantages. Even though it requires file handling to guarantee persistence, the data is stored on disk and only touched for a short period of time when the program is running. This minimizes the risk of data loss due to program or server crashes as most of the time I2P Observer is not running at all (and the data is protected by the very reliable file system) and even when the program crashes and loses data, only the current RouterInfoStatistic is affected.

Using the ObserverProperties class allows to easily adapt I2P Observer to new environments like running it on a new host or use another web server to present the statistic pages. Furthermore, it is easily extensible to collect more or other data available from the NetDB or run it more often, as discussed in Chapter 3.4.1.

### 3.3.2. Website

The website for I2P Observer is simple, but easily accessible and presents the essential information both visually for a fast overview as well as detailed in multiple tables. It has a clear structure and all necessary elements, including the "about" site which describes its function, the "help" page with explanation of terms and the "impressum" which states a contact address as well as used third party products. The separation of the statistic into daily and monthly ones ensures that there are not too many entries on each individual site. **Figure 3.3** shows a screenshot of the website.

Of course, there are possibilities to improve the website. Some ideas, which could not be implemented due to time limitations are collected in Chapter 3.4.2.

## 3.4. Discussion

### 3.4.1. Java Program

The Java program of I2P Observer works as intended, but could be optimized by refactoring its code as already mentioned previously. There are multiple features that would improve it, but which could not be implemented due to a lack of time. They may be added into a future version of I2P Observer and include:

- **Extension of collected data:** More data is available to be extracted from the I2P NetDB which can be added to I2P Observer, especially additional "caps" entries (like shared bandwidth, availability etc.) which specify certain group of routers.
- **Improve reliability of data:** Make a backup of files before importing them into I2P Observer and delete them once the program exits successfully to prevent loss of data in case of a program error. Currently the statistic for a whole day or month could be lost in this case.
- **Improve disk usage:** Reduce the disk space required by deleting old files. Serialized RouterInfoStatistic files older than today (for daily statistic) or this month are no longer required and can be deleted. HTML files older than a month (for daily statistics) or a year (monthly statistic) are no longer referenced on the website and can be deleted as well.
- **Use multiple instances to collect data:** To increase the accuracy of the data, multiple instances of the I2P router could be run and their data aggregated in I2P Observer. Ideally they are spread around to world to be independent from regional influences.

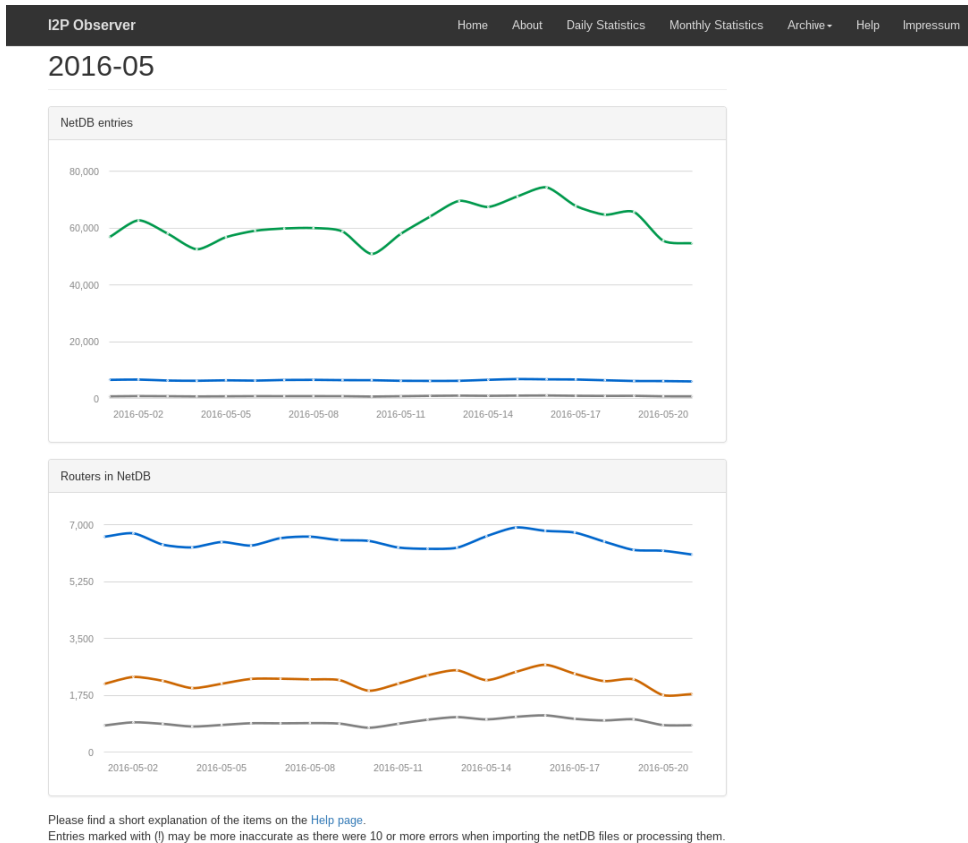


Figure 3.3.: Screenshot of the I2P Observer Website

- **Support for start parameters:** Implement support to use I2P Observer with start parameters, for example to enable debugging or to use different settings (like working directories, login credentials or targets).
- **Improve security:** I2P Observer currently uses an unencrypted FTP connection to upload the files to the web server. This is problematic in multiple ways as the credentials are sent in plain text and thus can be viewed by everyone monitoring the connection. Furthermore, the files could even be manipulated during transport which allows an adversary to include code (e.g. a tracking script that collect information about the visitors) if he has access to the connection. This would require the implementation of a FTP library into the Java code.
- **Check IPv6 support:** Verify if GeoLite supports IPv6 addresses in the current version and, if not, implement it. It is possible that the large amount of "Unknown" addresses in the Country table are IPv6 addresses which GeoLite currently can't resolve.

### 3.4.2. Website

The website displays the collected information in a structured and visually appealing way, but could be improved as well:

- **Present more data:** Include the possibility to display multiple statistics in the same chart. This would simplify the analysis for finding certain anomalies and influences, like time-based ones.
- **Better charts:** Implement other chart libraries that support more options
- **Present longer archives:** Instead of deleting old HTML files, a better overview page which links to the statistics could be implemented. Use a calendar-style overview page to link to the statistic for example.
- **Use encryption:** As already mentioned above, improve security by using encrypted connection for uploading files via FTP (SFTP / FTPS).

## 4. Adaptation of TOR Browser for I2P

The TOR Browser is a software bundle which offers an easy possibility for privacy-friendly browsing without the need to install or configure any additional software. It includes the TOR software together with a modified Firefox browser and can be downloaded from the TOR website [25]. Upon execution, it establishes a connection to the TOR network and launches the specially configured Firefox browser which offers a good privacy protection. This is achieved by routing all traffic through TOR and providing extensions like NoScript or Torbutton complemented by privacy-friendly default settings in the browser, like disabling JavaScript or blocking third-party cookies and tracking.

Similar to TOR, the I2P software provides proxy servers, which are used to route traffic into the I2P network, so configuring the TOR Browser to use I2P instead of TOR would allow an easy and privacy-friendly way to browse inside I2P without much configuration.

### 4.1. Methods

#### 4.1.1. Proxy Settings

Using I2P with the TOR Browser requires little change in configuration. The proxy settings need to be changed from the TOR client to the ones provided by I2P. As shown in **Table 4.1**, I2P does not offer a SOCKS proxy but is using HTTP and HTTPS ones, so the network settings of the browser needed to be changed by adding the I2P proxies and removing the one from TOR. Additionally, the extension Torbutton needs to be disabled as it also acts as security measure against unwanted manipulations of those settings.

Although this configuration works well, Torbutton also provides the strong privacy settings and options for the Tor Browser. Disabling it would therefor remove a big part of the browsers protection, so a possibility has to be found which allows the adjustment of the proxy settings without having to disable this extension.

#### 4.1.2. I2P with Torbutton

One possible solution to do this would be by using start parameters that tell the browser the proxy settings it should use. Analyzing the start scripts of TOR Browser (start-tor-browser and run-mozilla.sh in the folder /Browser/) did not show any possibility for such parameters. A quick online research indicated that the settings could be adjusted by manipulating the prefs.js file in the profile directory, but because this has the same effects as editing the settings inside the browser, the problem that Torbutton needs to be disabled for that to work persists. As this probably also applies for start parameters, a possibility to edit the setting of the extension was investigated alternatively. The about:config page, which provides advanced configuration settings, contains several parameters for Torbutton, including one for HTTP (extensions.torbutton.custom.http\_proxy and extensions.torbutton.custom.http\_port) and HTTPS proxies. However, changing them does also result in a connection error when trying to access the I2P network, so this attempt did not work as well. Further research showed that TOR Browser supports environment variables to manipulate its behavior. Even though they are not well documented and mainly rely on TOR\_SKIP\_LAUNCH to skip the start of TOR when an instance of it is already running, the variables needed for

	TOR	I2P	
Type	SOCKS v5	HTTP	HTTPS
Address	127.0.0.1	127.0.0.1	127.0.0.1
Port	9150	4444	4445

Table 4.1.: Proxy settings for TOR and I2P

the configuration of proxy settings, TOR\_HTTP\_HOST and TOR\_HTTP\_PORT and similar names for the HTTPS proxy were not hard to figure out and seemed to be adopted during the launch of the browser. Unfortunately, they only have effects on the Torbutton settings in about:config, so they do the same thing already tried earlier which turned out to not work.

A completely different approach found in an tutorial on the Eepsite TheTinHat[20] during research was to install an extension to TOR Browser called FoxyProxy, which handles different proxy settings according to the address of a website. This allows to access both the I2P and the TOR network with the same browser simultaneously. Installing this extension and importing the configuration file provided on the site worked as described, TOR Browser used the HTTP Proxy of I2P for http-requests to .i2p domains, the HTTPS one for https-requests to those and the TOR SOCKS proxy for all other queries. This configuration can also be made by hand if there is no trust towards the import file. The clear downside of this solution is that an additional extension needs to be installed which needs to be trusted.

### 4.1.3. Persistent Settings

To make the adaptation persistent so that it is available after an update of the TOR Browser, the use of an external profile was considered to be the most convenient way. This allows to save the settings of the browser including installed extensions outside the folder of TOR Browser and would not be affected when this folder is overwritten during an update. Additionally, the profile could be used to quickly access I2P with a new installation of TOR Browser without needing to configure it. Firefox allows to specify profiles with the argument "--profile <folder>". Unfortunately, the start script of TOR Browser only checks for some own parameters (detach, verbose, help, log, register-app and unregister-app) and discards other values before always starting the browser with the fix profile in /TorBrowser/Data/Browser/profile.default. Adjusting the script is of course possible, but the changes would be lost when upgrading the bundle which would overwrite the files.

A satisfiable solution is to extract the profile folder, store it separated from the browser bundle and use it to overwrite the default profile when installing a new version of the TOR Browser. This could be done either by hand or by creating a small script. When using the built-in update mechanism of the browser, the profile settings should not be changed and therefor making it not necessary to replace it. But even if this would happen, the correct settings can be restored by copying over the profile as described above. This works well as long as there are no needs for the TOR project to adjust settings of the profile for security or privacy reasons which would break the special proxy configuration.

## 4.2. Results

With the FoxyProxy extension, an easy and convenient way was found to use the I2P network with the strong privacy protection of TOR Browser. Furthermore, it allows the usage of I2P and TOR simultaneously which improves the comfort even more. Although there is no simple solution to provide this setup for new versions or installations of TOR Browser, the method described in the previous chapter is convenient enough as manually configuring the browser does not require much effort. This means that the goal of this short, optional task was well reached.

## 4.3. Discussion

Many improvements or other solutions are possible to reach the goal of using I2P with TOR Browser, the most useful being a fully configured extension for it or even a stand-alone bundle. On the other hand, these solutions bring the needs of maintaining them, like adjusting them for new program versions or changes both in TOR Browser and I2P, which would require quite some effort in the future.

The solution with the FoxyProxy extension could be improved by finding an easier way to install and configure it for new and updated versions of TOR Browser. If the way described in this report is used, only specific files from the profile folder could be identified and copied, making the amount of data needed to be handled much smaller. Also, this could be done with a script that is distributed alongside the files and, upon execution, copies the settings in the correct folder.

## 5. Test Network

For testing different attacks on I2P, a test network was created that provided multiple advantages. It consisted of a total of twelve I2P routers that were joined together to a small, internal I2P network. That amount of routers was considered to be big enough to represent the processes from I2P, so for example not all tunnels use the same routers, but still small enough to be manageable. Furthermore it allowed to manipulate each individual peer and monitor the whole traffic of the network at a central point, making it simple to track the impact of individual changes or attacks. It also ensured that the various tests and attacks do not affect the real I2P network, which was considered a very important condition.

### 5.1. Network Planning

#### 5.1.1. Basic Network Setup

The test network was built using Virtual Machines (VM) on a physical server running Ubuntu and libvirt. It consists of one management VM that is used to control the network and Internet access as well as to conduct various tests and measurements. To establish a segregated environment in which the attacks can be tested, ten VMs that act as I2P routers were set up and joined together to an internal I2P network, along with a client VM and a web server used for testing purposes. **Figure 5.1** shows the structure of the network.

To ensure that the tests do not affect the real I2P network, a strict separation between the internal network and the Internet was implemented with an iptables firewall on the Management VM.

#### 5.1.2. I2P Configuration

The I2P installations were configured using the I2P Router Console, a local website which can be accessed on port 7657 once the I2P software is running. It allows easy configuration of settings, gives an overview of the current I2P status and can also display a large amount of information about I2P. A screenshot of it is displayed in **Figure 2.1** in the previous chapter.

For the setup of the test network, knowledge about how I2P routers share peers is needed. As already described in the I2P overview (especially in Chapter 2.2.4), I2P stores information about routers in routerInfos in the NetDB. This includes the public key and the address (IP and port) of the router. The I2P software tries to determine the IP address (both IPv4 and IPv6) of the system automatically and uses a random port between 9000 and 31000 to provide an easy and user-friendly setup. It can also be configured manually from the network configuration if auto-detection should fail or a special configuration is needed like in this project. The following adjustments were made to the default settings:

- uPnP was disabled as automatic port forwarding was neither configured nor desired
- static IPv4 addresses were used in IP and TCP configuration
- IPv6 was disabled to simplify the setup

I2P does not allow the usage of private IP address spaces (like 10.0.0.0/24 which was used during the project) and provides a blocklist of those by default, which makes the setup of an internal network much harder. Even though the blocklist can be edited, the software insists on using public available IP addresses or hostnames in the network configuration to ensure other users are able to connect to the router. This provided some challenges as described later in this chapter.

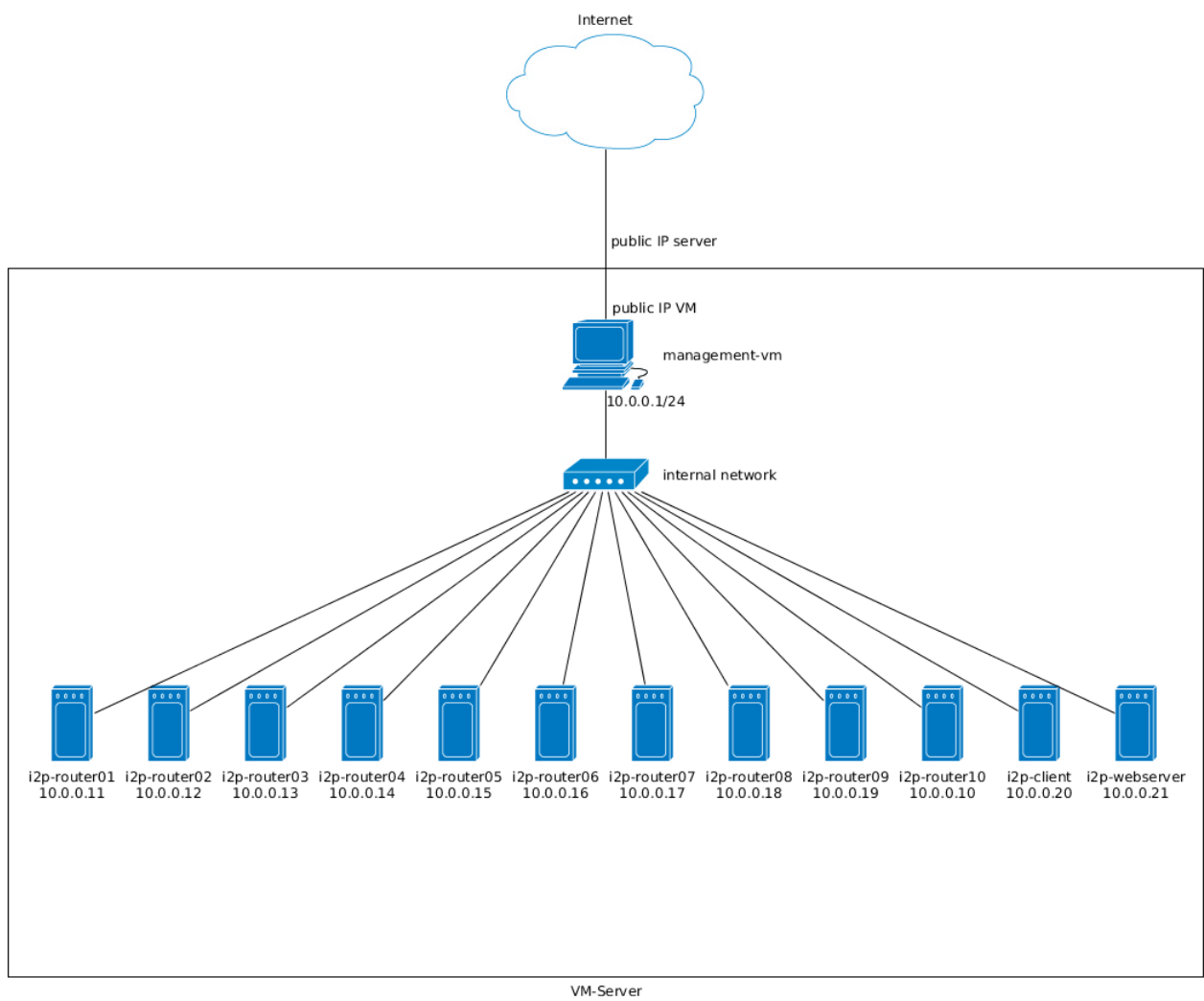


Figure 5.1.: Overview of the Test Network



## 5.2. Methods

### 5.2.1. Basic Network Setup

As first step, a plan of the virtual environment was created and the networks were configured on the hosting server, with one network allowing access to the Internet and an internal one with the IP address range 10.0.0.0/24 used for communication between the VMs only. Then, the Management VM was created and Debian Jessie with a desktop environment installed, before the network interfaces were configured and SSH access was enabled for remote access. Next was the creation of the basic VM template which was used to create a total of 12 VMs for the internal network via cloning: the 10 routers for the I2P network, an I2P client used for testing and an I2P web server which hosts a website that the client can access. The complete setup is shown in **Figure 5.1**. After adjusting IP addresses and hostnames on the machines and installing I2P, the basic setup of the network was complete and the configuration of the internal I2P network could begin. To separate the network from the Internet, a firewall was set up on the Management VM using iptables rules which only allow SSH connections to the internal network and it was configured to act as gateways for all other VMs. The complete rules of iptables are listed in Appendix A.3.

### 5.2.2. Connection to VMs

To connect to the Virtual Machines, SSH with public key authentication was used. Due to the setup with complete isolation of the internal network, the Management VM (which was the only VM with access to the Internet) was used as jump host. To provide the possibility to change the configuration of the I2P software, port 7657 which allows access to the I2P Router Console was forwarded via a SSH tunnel. This setup allowed to easily configure all VMs remotely from one machine by using different ports for every tunnel. Additionally, port 4444 and 4445 which provides access to the built-in I2P proxies was forwarded from the i2p-client VM the same way. An example of the SSH configuration can be found in the Appendix in Section A.1.

To be able to control all relays at once, a shell script was created that executes a given command on every of those systems. The script's code can be found in Appendix A.2.

### 5.2.3. I2P Configuration

The tricky part of the setup was to connect the I2P routers in the network without allowing them to access the Internet. In the default configuration, I2P tries to determine the IP address automatically, which obviously failed due to the lack of Internet access. This resulted in the own routerInfo in the local NetDB to not contain any address for that router. As this routerInfo was later used to introduce the router to the other ones in the internal I2P network by distributing it, the address needed to be set manually in the network configuration of I2P. As mentioned in Chapter 5.1.2, the I2P software only accepts a publicly accessible IP address in its configuration. The first approach to solve this issue was to setup a DNS server on the Management VM, configuring it to resolve certain domains (relay01.com to relay10.com) to some public IPs and use this domains as hostnames in the network configuration of I2P. Once the address is set and the routerInfo with it was created, the DNS server is then changed to resolve the domains to the correct internal IPs. This strategy worked well for the configuration part, but after distributing the routerInfos across all routers they refused to connect to each other, probably due to some restrictions implemented in the I2P software.

Since using a DNS server turned out to not be an option, no other alternative was found than to configure the I2P routers with real public IP addresses. To maintain the setup with the isolated network, some addresses were chosen at will (123.123.123.10 - 123.123.123.21) and the gateway was configured to reroute them to the actual, internal addresses (10.0.0.10 - 10.0.0.21). The exact setup is described below.

- The I2P routers are configured to use public IPs (e.g. router01 uses the IP address 123.123.123.11, router02 uses 123.123.123.12)
- router02 sends a packet to 123.123.123.11 (router01)
- The gateway uses an iptables PREROUTING rule to reroute the packet to 10.0.0.11 (internal IP of router01) by changing the destination IP address, so called DNAT

- Additionally, the gateway uses a POSTROUTING rule to change the source IP address from the internal one of router02 (10.0.0.12) to 123.123.123.12, called SNAT
- Similar rules are used for the reply (external IP of the destination is exchanged with the actual internal one, the internal IP of the source is replaced with the fake external one)
- This tricks the routers to act as if they were communicating with each other over public IPs

The complete iptables rules can be found in Appendix A.3.

Once all ten routers, the client and web server were set up with their fake public IP addresses and the DNAT and SNAT rules were enabled on the Management VM, the routerInfo file of each router was copied from the local NetDB folder, merged into a zip archive and then distributed to each machine with the reseeding function of I2P. After a restart of the I2P software to accelerate the reload of the NetDB and a verification of the settings, the routers started to build tunnels and communicate with each other. To reduce network traffic caused by the routers, the reseed feature was disabled by entering 0.0.0.0 as only source from which each router should query new peers.

As no router seemed to have volunteered to act as Floodfill Router at the beginning, which in theory should have happened automatically, two of them, i2p-router01 and i2p-router02, were forced to do so via the configuration. Otherwise, no leaseSets could be exchanged and therefore the I2P client VM could not access the website of the I2P web server.

To verify the proper setup of the internal I2P network, the web server plugin on the i2p-webserver VM was enabled to host an Eepsite and the accessibility of it was tested from the i2p-client VM. It took some time until the leaseSet of the web server was published in the NetDB, but once that happened, the site could be accessed without any problems.

## 5.3. Results

### 5.3.1. Basic Network Setup

The basic setup could be achieved without major noteworthy issues. Completing the setup of the systems and adjusting settings on the cloned relay VMs took the most time for this step. Afterwards, the hardware-related setup was complete and allowed to start with the software configuration.

### 5.3.2. Connection to VMs

Providing access to all VMs via SSH was a simple task. Even the more difficult part, configuring all SSH tunnels to access the I2P Router consoles as well as the I2P proxy servers on i2p-client was done without any problems. The shell script proved to be very handy when configuring or maintaining the relays.

### 5.3.3. I2P Configuration

Once the solution to use DNAT and SNAT rules in iptables was found, so public IP addresses could be used inside the network as described above, collecting the routerInfos of all routers and distributing them via the reseeding function of I2P was easy. Besides minor adjustments in the network configuration, no further setup was needed for the internal I2P test network to work properly.

## **5.4. Discussion**

### **5.4.1. Basic Network Setup**

The test network turned out to be fully functional during the evaluation of possible attacks. It could be improved by providing graphical environments for every VM, as handling all of them via SSH was found to be extensive in some cases, especially when adjusting the configuration of multiple routers individually during the evaluation of an attack. Other than that, no major issues were experienced during the use of the network which could be improved.

### **5.4.2. Connection to VMs**

No problem or possible improvements were found for the SSH configuration. The command script for the relays could be improved by adding a more convenient way to execute commands with root permissions (sudo) which is needed from time to time.

### **5.4.3. I2P Configuration**

Finding a way to configure the I2P relays to use IP addresses without the need of DNAT and SNAT would make traffic capturing much more convenient. With the current setup, all packets are captured twice when using the management VM to monitor the traffic inside the test network, once from the internal address of the source to the public one of the destination and once from the public source address to the local destination.



## 6. Attacks on I2P

As already mentioned earlier, getting to understand the processes inside I2P and evaluating its security was the main motivation for this thesis. Apart from creating the introduction in Chapter 2, understanding previously published attacks on I2P and analyzing possible weaknesses offered a large source of knowledge. Especially researching possibilities to forge NetDB entries as specified in Section 6.1.1, which resulted in the Clickjacking attack described in Chapter 6.1.2, provided a big amount of know-how about I2P.

Four different approaches were chosen to analyze possible security flaws inside I2P, which are described in this chapter.

### 6.1. Methods

#### 6.1.1. Forgery of leaseSets and routerInfos

The main goal of this approach was to get the I2P router of a victim to communicate with a peer controlled by the attacker instead of the legit one. Multiple possibilities were evaluated for their probability for success.

The first, most obvious approach would be to forge the leaseSet entry in the NetDB itself, as it contains the contact information (inbound gateway and tunnel-ID) of a destination. However, as the destination is the SHA256 hash of the services identity (ElGamal key, signing key and certificate), it is unfeasible to compute a different set of keys which result in the same hash value as this can only be done via bruteforcing. In case it would be possible to create a collision for the identity, it would be trivial to create a leaseSet with own tunnel information and publish it in a way that this one is retrieved by the victim instead of the one from the service that should be impersonated. This could be done by either deploying floodfill routers under control of the attacker which always propagate his leaseSet (similar to the attack described by Egger et al. [21]) or by simply publishing it shortly after the legit service to overwrite it, which for the NetDB looks like an ordinary update.

But as long as the collision resistance and one-way property of SHA256 holds and it is properly implemented, there seems to be no possibility to publish a leaseSet for another service. Furthermore, manipulating a published leaseSet appears to be not possible as well as it contains a signature which can't be forged unless the private signing key is known or ED25519 signatures, which are used for this purpose, can be forged. Using another signing key and publish that one together with the new destination is prevented by the SHA256 hash, which would change as discussed above.

As forging or manipulating entries in the NetDB seems to be very hard, if not impossible, the more promising approach would be to target I2Ps address book as it contains the mapping from human readable domain names to destinations. If it is possible to add or alter entries in it, impersonating a service becomes rather easy by setting up a new one and make the victim access it via its address book. There seem to be different possibilities for a manipulation which demand varied requirements from the attacker:

- **Gain control over the I2P project server:** The server which hosts `i2p-projekt.i2p` is used for the synchronization of the default subscriptions. If an attacker can take control over the server (physically or via impersonation), he can manipulate all entries and impersonate all services listed there. As all I2P routers are subscribed to it per default, this possibility covers a very wide scope. However, this attack would not allow to impersonate services stored in the private or master part of the victims address book and would probably be hard to implement, as these servers are supposed to be well protected.
- **Gain control over another subscription service:** Similar to the previous scenario, controlling one of the other subscription services in I2P would allow to easily manipulate address book entries for a larger amount of users. However, they need to subscribe to this service first and the same restrictions mentioned earlier apply.

- **Gain control over a Jump Service:** Another possibility is to take over control of one (or more) of the Jump Services in I2P, which resolve unknown domains into their destination. This can be used to redirect users who do not yet have an entry for a specific site in their address book to any server. I2P's feature which offers to add the newly learned destination to the address book can be very convenient in this scenario. When the victim chooses to do so, the Jump Service needs to be controlled (or impersonated) only during one request to permanently send the user to the faked service. This scenario requires the user to trust the Jump Service, which seems feasible depending on the security awareness of the user.
- **Add Address Book Entries with I2P Addresshelper:** The I2P Addresshelper is used by Jump Services to add the destination for a resolved domain to the address book. This is done by redirecting the client to a specific URL of the form `http://domain.i2p/?i2paddresshelper=<destination>`, which triggers an internal website of the I2P router that allows the user to add an entry for that domain to the address book. If the user clicks on one of the buttons to add it to one of the address books, a GET request is sent to `http://proxy.i2p/add?<params>` with the hostname, the destination, a nonce (which appears to actually be a session or time-based ID), the part of the book it should be added and an URL as parameters. As the nonce can not be known from external nor read, it prevents adding new entries via CSRF (Cross Site Request Forgery). However, if it is possible to acquire that nonce, it can be used to add additional entries to the address book by simply sending a GET request to an URL.
- **Add Address Book Entries with Clickjacking:** This approach is similar to the previous step, but tricks the victim into adding the address book entry himself. It can be done by hiding the I2P Addresshelper page in an opaque iframe and design another page which persuades the victim to click on an area of the page which overlaps the button on the Addresshelper site. Depending on the location where the user clicks, any entry to any part of the address book can be added. This kind of attack is known as "Clickjacking" and is described in the following section in detail.
- **Manipulate Address Book via CSRF:** As the user can add, edit and remove entries in the address book via the I2P GUI, the possibility to do the same via CSRF was evaluated. All those operations are done via POST parameters and use a serial for security, which acts as nonce and is transmitted with the page content. Parsing this serial with JavaScript may be possible and would allow to manipulate the address book, but was not further pursued in favor of the Clickjacking attack.

### 6.1.2. Forge Addressbook Entries with Clickjacking

As described in the previous chapter, Clickjacking can be used to add entries to a victims address book. This is done by hiding the page of the I2P Addresshelper in an opaque iframe and constructing a web page that tricks the user into clicking on the button which is invisible for him. Usable on every Eepsite, this attack can add any arbitrary entry to the address book of the user as long as there is no entry for the hostname in it yet. Otherwise, I2P will display a warning about conflicting destinations, described more in detail in the following attack.

A more complex approach would be to mimic the usual process an user expects when clicking on a link. A very simple example is displayed in **Figure 6.1**. This attack could be reproduced without problems, although certain restrictions apply as described below. The HTML code of the example page is shown in Appendix B.2.1.

The biggest issue when exploring the attack was that I2P seems to load the I2P Addresshelper page only once. If it then gets reloaded or loaded for the same address a second time again, the page simply does not load at all, making this attack unusable until I2P is restarted. However, this mostly affects only the preparation of the attack like building the malicious website, as during a real attack it is sufficient if the victim loads the page only once.

Another possibility that prevents this attack is that the user already has an entry of the hostname in his address book. In this case, I2P warns him that the destination it received from the URL differs from this entry and offers the victim to choose which one he wants to visit, as displayed in **Figure 6.2**. Although this does not allow to manipulate the entry in the address book, it can be used as link to redirect the victim to a malicious site, especially as it is located in the same spot as the buttons used to add entries to the address book.

A smaller issue is that I2P automatically loads the Eepsite once the entry has been added to the address book. This could raise suspicion when the site loads unexpectedly, but may be circumvented by letting the current page reload once the user clicked on one of the buttons (which will not load the Addresshelper page again as discussed above) or by designing the button on the attack page in order to be perceived as link.



Figure 6.1.: Example of Clickjacking: A page with visible iframe (left) and how the user sees it (right)

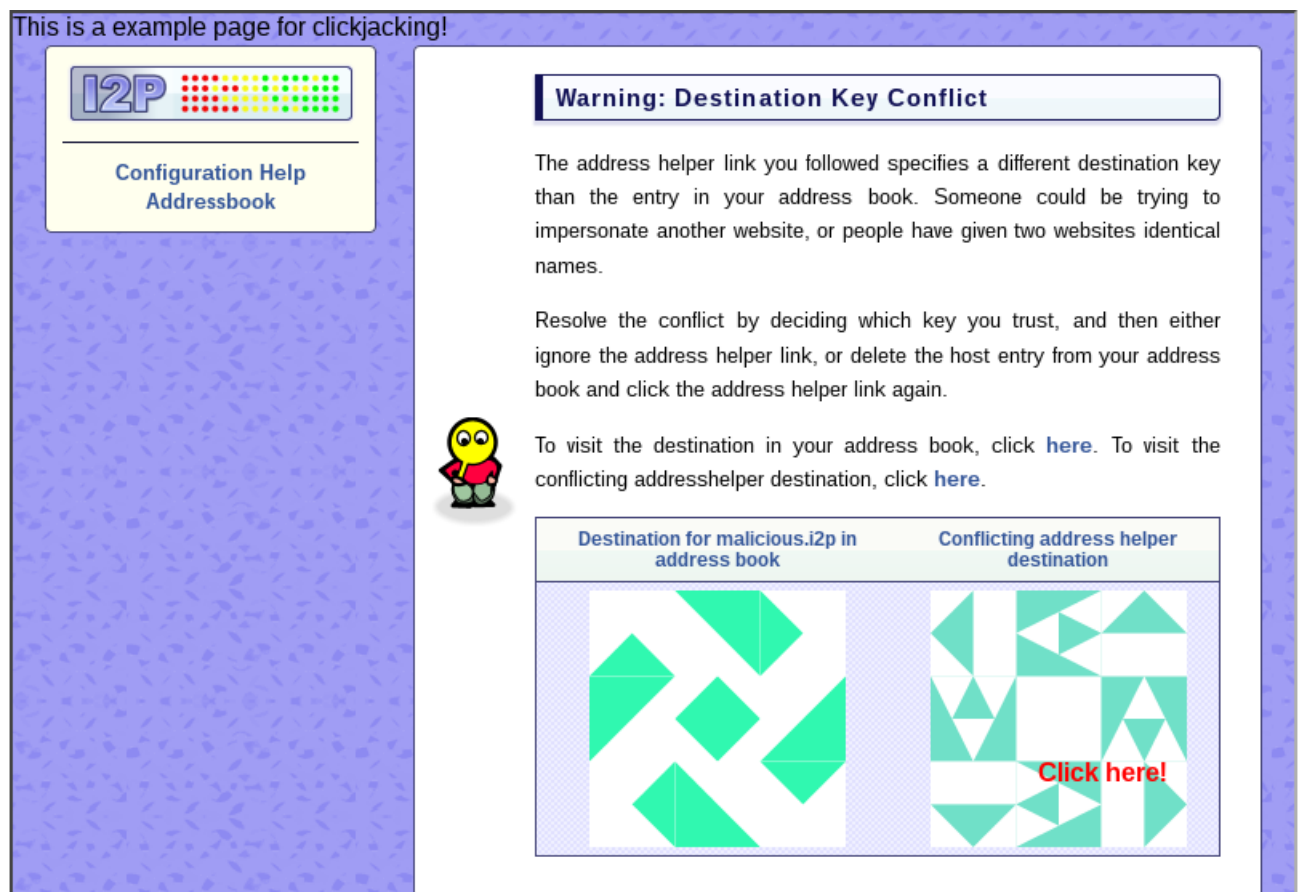


Figure 6.2.: Warning displayed by I2P when trying to add a conflicting key to the address book

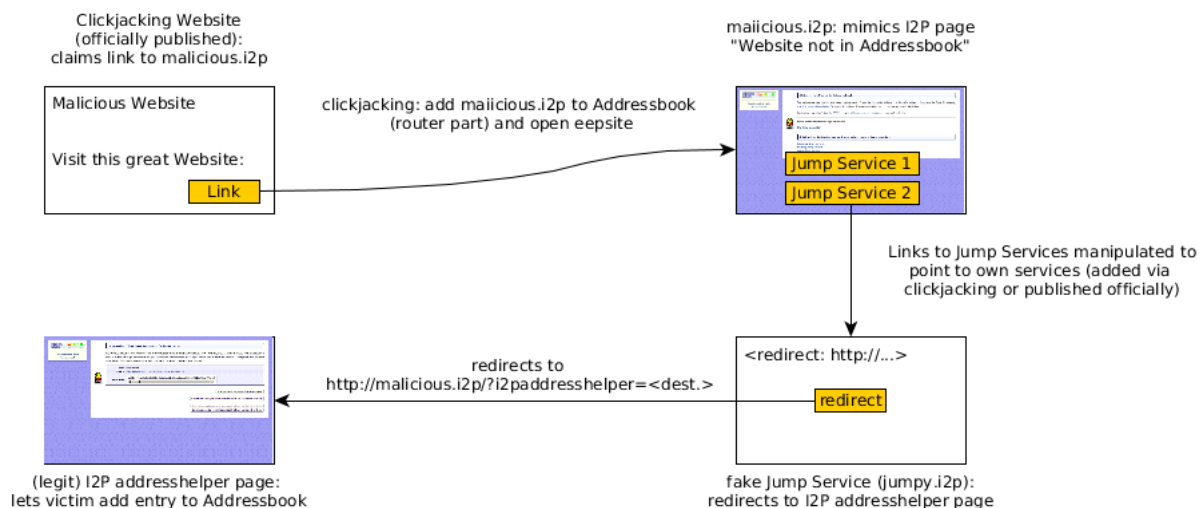


Figure 6.3.: Concept of Clickjacking Attack to add arbitrary entries to an address book

Over all, the Clickjacking attack might be successfully used in practice. The most naturally looking scenario would be to embed it into a site which presents a link to the Eepsite the attacker wants to impersonate. If the victim does not have an entry for that site in his address book, it is added and the user will always visit the malicious site in the future. If this entry is present, he still visits the malicious site this time and every time he clicks on the link. However, further research is needed to create a proof-of-concept as a possibility to load the new site outside the (invisible) iframe is required. Otherwise, the victim would not see the site. Nevertheless, loading the site and using scripts to try to identify the victim may be enough, depending on the attack scenario.

To make this attack even more subtle, it could use 2 additional Eepsites, one that displays a faked "Website not found in Addressbook" page where all Jump Service links point to the same faked Jump Service (or even imitate the 4 default Jump Services with similar addresses). The faked Jump Service then simply redirects to the I2P Addresshelper page with the destination of the malicious Eepsite, getting the victim to add the address book entry himself. The first Eepsite (the faked error page) and the actual malicious site need to have similar addresses to not raise suspicions, but not the same as I2P would displaying a warning message about conflicting keys in this case. This can be achieved by the same way Phishing works, like replacing a lowercase L with an uppercase i. The whole process is illustrated in **Figure 6.3**.

Alternatively the site could simply act conspiratorially by claiming that it redirects to the Addresshelper page with the alleged correct destination and let the user add the entry himself while letting this process look common.

As mentioned before, this attack only works when the user does not already have an entry in his address book as in this case he may not expect to see the error page and the Addresshelper page generated from the Jump Service would display the warning about a conflicting key, making the victim very suspicious.

### 6.1.3. Identification of I2P Traffic

Planning the concept to identify I2P traffic resulted in two basic possibilities, to collate it based on its target being a known I2P router and to find specific patterns that can be used to identify individual I2P packets.

To find similarities in I2P packets, the traffic in the test network was captured during three different states: when starting the I2P router, after running it for a while and during the shutdown process. The traffic dumps were then analyzed to find specific patterns or packet properties which allow to identify them. As all I2P communication is encrypted, only the information up to OSI layer 4 are visible and can be analyzed without the effort of decrypting the traffic (if that is possible at all). Due to that, I2P specific patterns of the packet could not be detected. Over all, not many characteristics could be found to distinguish I2P packets from regular ones, except

- **IP Address:** The IP addresses of all routers could be extracted from the NetDB (similar to what I2P Observer does) to at least identify communication with hosts which participate in the I2P network. Furthermore, DNS resolves of the domains and access to the servers used for reseeding, as described in Section 2.2.4, can further indicate that one host in the network is using I2P.



- **Port numbers:** I2P uses random port numbers greater than 9000. While this randomness does not allow to directly identify packets from I2P, it can be extracted per host from the NetDB. Furthermore, if a host is communicating with many different peers on different high ports, this can be considered unusual and could be an indication that I2P is running.
- **Flags:** Many TCP packets of I2P use the PUSH flag to tell the recipient to forward the packet as soon as he receives it. While this is no unique characteristic of I2P, it can be used to further harden a suspicion.

In conclusion, detecting I2P traffic inside a network is hard without any preparation, as individual packets are encrypted and do not have an unique characteristic. However, it can be done by extracting information from the NetDB and then matching IP address and port number of the destination against it.

#### 6.1.4. Requirements for secure Garlic Routing

Garlic Routing (see Chapter 2.2.7) bundles messages to prevent Timing Attacks. The goal of this approach was to identify how much traffic is needed so this technique works and hides individual packets, so an attacker who monitors all connections of one router can not follow them. It is possible that it could fail when there is very low traffic (like one packet every 10 seconds).

This scenario was implemented by using the separated I2P test network described in Chapter 5 and start sending very few packets between nodes (e.g. one ping request every minute from one node to another) while monitoring network traffic and decreasing the time between packets and increasing the amount of nodes interacting until individual packages can no longer be distinguished.

To gather facts about the traffic I2P creates when no user content is routed, all packets that were sent and received by a router in the test network were monitored on two I2P routers using tcpdump. This showed that they handled about 90 to 100 packets per minute for maintenance of the I2P network, like tunnel building or NetDB lookups. Even though there are peaks with 10 or more packets per seconds and intervals up to 10 seconds where no packets are transmitted, not even a single packet sent over I2P could be traced as it could not be distinguished from the traffic created by the I2P router itself. So even the base traffic was found to be more than sufficient to hide meta data of packets with Garlic Routing.

#### 6.1.5. De-anonymization via High Traffic Attacks

The concept of this part was to recap a High Traffic Attack to find the recipient in the I2P network.

This attack works by sending a high amount of packets towards the target while observing the complete network. If every connection can be monitored, it should be possible to follow the path of the packets from router to router until the destination is reached and can be identified.

To perform this attack in the test network, a connection to the i2p-webserver was established while all network traffic was captured using tcpdump on the Management VM. The capturing was started by monitoring the normal base traffic of the internal I2P network to get a comparison for later measurements. Then, after 15 seconds, the website was loaded once. After another 15 seconds, it was reloaded twice before waiting for 30 seconds and then generating heavy traffic load by reloading the site as soon as it finished loading, using Ctrl + F5 to circumvent the cache. This created enough data which then was analyzed using Wireshark.

The analysis showed that under those circumstances, the single loading of the web page was sufficient to reconstruct the chain of participating routers and therefor de-anonymizing the server. This was done in two steps. First, the statistics about conversations in Wireshark (found under Statistics -> Conversations) was used for the IPv4 addresses. When sorting the list by amount of packets, it became quite clear which peers communicated with each other during the loading of the web page as they had a much higher packet count than those not involved (for the test it was 189 - 755 packets against 50 and less for routers not participating). This information was then mapped to a graph. In the next step, the traffic was analyzed chronologically and the edges of the graph marked in the order of usage. This showed all hops between client and server pretty clearly. For the reverse path, the fact that in this connection 3 hops were used twice made it a bit more challenging to find the correct order of participants, but once the first part of the connection was resolved using the method described above, the correct sequence of routers could be determined. The reconstructed connection order can be seen in **Figure 6.4**.

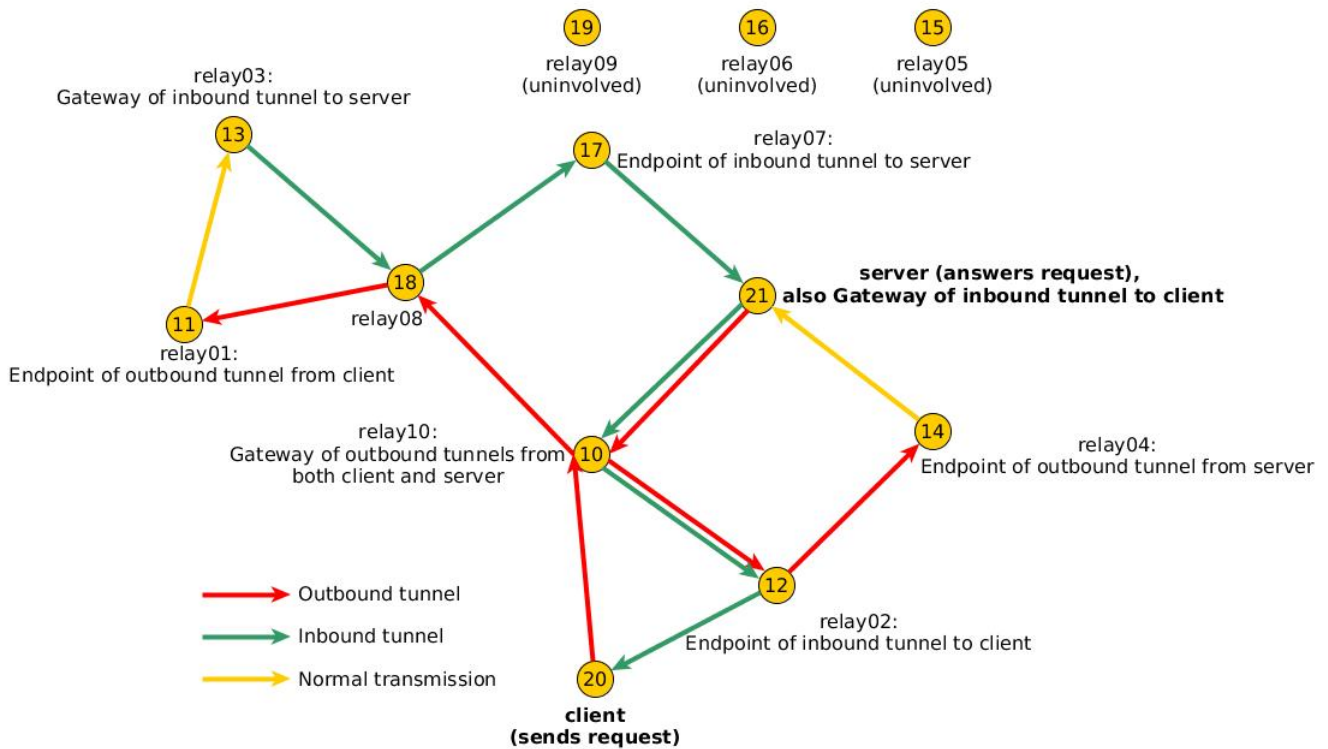


Figure 6.4.: Graph of reconstructed traffic path, reenacted with the High Traffic Attack

## 6.2. Results

### 6.2.1. Forgery of leaseSets and routerInfos

The evaluation showed that it is unfeasible to forge or modify leaseSets or routerInfos as long as there is no weakness in the cryptographic procedures or their implementation. Even if those could be proven insecure in the future, it should be easy to fix bugs or change the cryptography used so that I2P is no longer vulnerable.

Manipulations to the address book seem to be a bigger threat, depending on the adversary and user. If the adversary is powerful enough to gain control over certain servers used in I2P for synchronizing the host lists or as Jump Service, he can impersonate a high amount of websites towards many to nearly all users by manipulating the correlation between hostname and destination. This scenario seems rather unlikely and an adversary with this possibilities has probably more effective attack vectors (like manipulating the source code).

Changing entries in the individual address book of an user or adding arbitrary entries seems to be a more feasible scenario. The Clickjacking Attack showed that it is possible to trick the user into adding entries with the wrong destination under certain conditions and using JavaScript to manipulate the address book with CSRF may be possible as well.

### 6.2.2. Forge Addressbook Entries with Clickjacking

Adding arbitrary entries to the users address book via Clickjacking seems to be quite easy if there is no entry for the same hostname in it and can be done on every Eepsite as shown previously. To make the process more natural, the more complex approach with the faked error page and Jump Service can be used. If the user however already has an entry in the address book for the hostname, it gets much harder as every Addresshelper page used during this attack would show warnings about conflicting keys. The best possibility here would be to use Clickjacking to let the victim visit the conflicting destination.

All of this possibilities still rely on more research to find a way that lets the new page load outside of the iframe and only works inside a browser configured for I2P, as an I2P address needs to be accessed.

### 6.2.3. Identification of I2P Traffic

It is possible to identify I2P packets in the traffic flow of a network, but that requires some preparations. If it succeeds to extract all information about I2P routers from the NetDB, the IP address of every router and the corresponding port that is used by I2P is known and can be matched against network traffic to identify I2P connections. Gathering the data of all routers is not trivial, but necessary to prevent false negatives.

Without the possibility of collecting that data, it seems unlikely to define specific characteristics which can be used to identify I2P traffic. The only hint is a larger amount of connections to different high port numbers, which can raise suspicion, but can not be related to I2P specifically.

### 6.2.4. Requirements for secure Garlic Routing

As individual packets can not be distinguished from the always present traffic created by the I2P router itself for network management, the conclusion to the initial question of how many traffic is needed so Garlic Encryption can obfuscate individual packets is: None.

Taking into consideration that the observations were done in the test network which consists of only 12 participants, the amount of traffic generated in the real I2P network by multiple thousands of routers is much bigger. This makes it even less likely to be able to follow a single packet through the network. Furthermore, I2P is used for filesharing with a modified BitTorrent client which should generate even more traffic in which other connections can be hidden.

Even though the recap of the High Traffic Attack in Chapter 6.1.5 seems to contradict the findings in this section that individual packets can not be identified, this attack relies on the fact that the analysis of all network traffic is used to identify participating routers based on the amount of packets exchanged during the test. As such statistics can not be done by monitoring the connections of a single router, this approach can not be used to target individual packets inside Garlic Routing.

### 6.2.5. De-anonymization via High Traffic Attacks

It was shown that a High Traffic Attack can succeed in I2P, which is little surprise as there is no real protection against this kind of attack. However, there are multiple factors to take into consideration. First of all, the attack is outside of the attacker scope of I2P against which the software should protect the user, which assumes that an attacker only has the possibility to observe parts of the network and therefor can not follow the entire path of the packets. While this assumption can be considered feasible, it is yet to be shown that these limitation can not be bypassed by using Timing Attacks (counting how many packets pass the border routers of the observed part of the network to identify traffic flows) when being able to monitor a big part of the network. Furthermore, the destination address of the traffic, the web server, was known during the test and therefor the start and endpoint of all tunnels could be determined. While the default configuration of I2P can be used to count the amount of hops for each tunnel (which is 3 hops per direction) to deviate the position of the server, the possibility to change their length means that such assumptions can not be considered highly accurate. Also, this attack was conducted in the test network with a small amount of routers and very little other traffic. If adapted to the productive I2P network, which has much more clients and traffic, a much higher amount of bandwidth is needed for this attack to succeed.

## 6.3. Discussion

### 6.3.1. Forgery of leaseSets and routerInfos

As already discussed earlier, forging or manipulating entries in the NetDB seems very unlikely supposing there is no flaw in the cryptographic methods or their implementation.

Manipulating the address book however seems feasible under specific conditions. The capabilities to take over important servers is a strong assumption, but may be possible for potent attackers like major state agencies. To expect specific behavior on the victims side, like not checking URLs and site names carefully can be valid depending

on the constraints of the attack and the users awareness. While using this attacks on a small service which is only used by a few people that are very careful and get suspicious quickly may be hard and could fail, successfully targeting larger sites with users which are less cautious, like a Darknet marketplace seems possible.

### **6.3.2. Forge Addressbook Entries with Clickjacking**

As the potential of a Clickjacking attack depends on multiple constraints, more research is needed to develop a successful proof-of-concept. There are two big challenges that still need to be solved to turn this approach into a powerful attack. One is to get websites to be loaded outside of the iframe, so it looks like clicking a legit link when the user actually clicks on a button on the hidden Addresshelper page. The other one is a possibility to differentiate if the victim already has an entry for the hostname in his address book or not. If this can be done, the attack can be targeted exactly to either add the arbitrary entry using the faked error page and Jump Service or persuading the user to visit the Eepsite using the conflicting destination. It might be possible to solve these problems with either JavaScript or HTML5 code, but more research in this direction is needed.

While researching this topic, one observation which seems to make a Clickjacking Attack like the one described in Section 6.1.2 much easier was that the buttons on the Addresshelper page and the link to visit the conflicting destination on the warning page are inside the same area of the websites. This allows the usage of a single button to let the victim access the website of the attacker, regardless of the circumstances.

A simple change to I2P, which could make the exploitation of such an attack more difficult, would be to change the layout of one of these sites, so there is no overlapping area for adding entries to the address book and visiting a conflicting destination.

### **6.3.3. Identification of I2P Traffic**

The way of extracting information about all I2P routers from the NetDB promises to allow a quite accurate identification of I2P traffic. It appears feasible that a setup to do this can be build, like combining the information from multiple floodfill routers.

Even though in this case it would be possible to hide the traffic by using I2P peers that are not published in the NetDB, but instead building an own I2P network, this would require big expenses and contradicts the purpose of I2P, as using only self-owned routers does not provide anonymity, but can be used to obscure the path of a connection at most. To achieve this goal, there are probably better and far easier possibilities available.

The identification of individual I2P packets was found to be hard as not much distinct characteristics were found. Because only a small amount of the research efforts were used for this part, it may be possible to relate unique patterns to I2P packets by conducting more research and analyzing them in depth.

### **6.3.4. Requirements for secure Garlic Routing**

The evaluation showed that the traffic created by I2P itself to maintain the network is sufficient to hide individual packets in it and therefor their meta data. As there are many more peers in the productive I2P network which also generates more traffic, packets should be hidden even better in this case. However, it is possible that there is a specific effect created by this amount of routers that falsify this finding, like the fact that not all routers are communicating with each other. A small amount of packets could stand out if they are too few for the typical inter-router connection. More research is required to examine if specific patterns can be identified in the maintenance traffic of I2P. While this relates to the question if I2P traffic can be identified in general, it is even more refined to find user-generated traffic.

### **6.3.5. De-anonymization via High Traffic Attacks**

While the High Traffic Attack definitely is an impressive demonstration and seems to be impossible to defend against, it is also out of the scope of I2Ps attack scenarios as already discussed. An adversary who can observe the whole network seems to be an unrealistic assumption, as even the major state agencies who can be assumed to monitor most of traffic of the Internet probably don't have this ability. However, it seems not totally unfeasible that

some of them can observe a part of it large enough to correlate I2P traffic inside the unknown part with Timing Attacks. Defending against such an attack is difficult because the simple transmission of packets is sufficient for it to succeed, even if the target would simple discard all of them. The only possibility would be that the routers inside the network detect such an attack and refuse to forward all following traffic from the sender. However, finding a rule that would effectively prevent this kind of attack without having an impact on legitimate connections seems to be nearly impossible as routers participating in a tunnel can't differentiate between packets. The goal of further research could be to identify how much of the network needs to be monitored to be able to conduct such an attack.



## 7. Conclusion / Results

Although the efforts needed to complete some of the goals of this theses were bigger than expected, it was possible to complete all of them. The introduction to I2Ps mode of operations was an easy task over all, although it was challenging finding the adequate depth of technical explanation at times. Additionally, the documentation published on the I2P website is inconsistent regarding specific terms at some points and often explains mechanisms of older versions of the software, where it is not always clear if this still applies to the current version.

Creating I2P Observer was an interesting task, especially as the whole program needed to be developed from scratch. Once the local folder of the NetDB was chosen as source for the information, the development was quite straight forward and led to a solid base version which offers possibilities for further expansion.

Configuring TOR Browser to be usable for browsing I2P proved to be uncomplicated. Using an extension to handle the settings of the proxy servers based on domain name clearly has its advantages and disadvantages, especially as it means relying on third party code and allows no automatically configuration for newly installed browsers. As this part was only seen as little experiment out of own interest, this was considered to be a well enough solution.

The setup of the test network was an ordinary task which only needed some time for completion. The biggest challenge was to circumvent I2Ps protection which restricts the usage of internal IP addresses, but a solution was found by rerouting the packages using iptables on the management VM.

For analyzing possible attacks on I2P, much more time was needed than expected, especially as many open questions related to them are not documented on the website of I2P but would require an in-depth analysis of the source code to answer them, which was out of scope. Nevertheless, it was possible to find a potential attack on the address book of an user which is a good result for the limited time available for this part. On a downside, this evaluation took the vast amount of time, so other approaches could only been analyzed briefly.

### 7.1. Future Work

Some findings of this thesis are worth looking further into. One big part is I2P Observer which can be improved by implementing additional features as already described in Section 3.4.

The attacks on I2P in this chapter can be used as basis for further research, especially on the following topics:

- Can JavaScript be used for a CSRF attack on the address book as described in subsection 6.1.1?
- Creation of a proof-of-concept for the Clickjacking attack by bypassing the current limitations discussed in chapter 6.3.2
- Verification of the ability to identify I2P traffic by writing a program that collects router information from the NetDB and creates a list with all IP addresses and ports which is then used to actually find I2P packets in a network stream.
- Prove that Garlic Routing works in the I2P network by conducting further research which shows that maintenance traffic does not generate an identifiable pattern.
- Research if a High Traffic Attack can be done when only being able to observe parts of the network by using timing attacks to correlate traffic and if so, under which circumstances.






# Declaration of primary authorship

I hereby confirm that I have written this thesis independently and without using other sources and resources than those specified in the bibliography. All text passages which were not written by me are marked as quotations and provided with the exact indication of its origin.

Place, Date: Biel, 16.06.2016

Last Name, First Name: Müller Jens

A handwritten signature in black ink, appearing to read 'Jens Müller', is written over a light gray rectangular background.

Signature: .....



# Glossary

**address book** Part of I2P which is used to resolve human readable hostnames to I2P destinations, similar to how DNS works ("domain name -> IP address" is the equivalent to "hostname -> destination" in I2P). See Section 2.2.5..

**Clickjacking** Attack where the content of a website is hidden inside an invisible iframe. The victim is perceived to click on a specific spot which activates a link or button located inside the iframe..

**Clove** One of multiple messages for the same recipient that are combined by Garlic Routing.

**destination** Contact information for a specific service needed to communicate with it. Contains the address of the Gateway, the Tunnel-ID and the Identity of the service..

**Eepsite** Website that is hosted and accessible from within the I2P network..

**Floodfill Router** Router that is part of the decentralized NetDB (see Chapter 2.2.4).

**Garlic Routing** Concept of I2P to use layered encryption, bundling messages for the same recipient or simply encrypt messages. Detailed explanation in Section 2.2.7.

**High Traffic Attack** Sending a large amount of data towards the recipient while observing the network, following the path of the many packets to deanonymize the recipient, similar to a Timing Attack.

**I2NP** I2NP messages contain one or more whole or partial messages with delivery instructions intended for one specific router.

**I2P Tunnel** Set of routers which are used to forward messages. The default setup consists of 3 involved devices: Gateway, Participant and Endpoint. Tunnels can be outbound (used to send messages) or inbound (used to receive messages).

**Identity** SHA256 hash of the ElGamal key, the signing key and the certificate.

**leaseSet** NetDB entry containing information about a specific user / service (keys and certificates).

**Meta Data** Data that is created by every communication process and can not be hidden by encryption. Typical examples are IP addresses or phone numbers.

**NetDB** Distributed database which contains all information needed to contact a router or service.

**routerInfo** NetDB entry containing information about a specific router (keys and certificates).

**Timing Attack** Finding the path of a message and its recipient by timing the amount of packets that go into a router and leaving it. If one observes that 4 packets are sent to the router and shortly afterwards this router sends 4 packets with the same size (or a size reduced by a fixed amount due to encapsulation) to another one, one can predict that it has forwarded his packets to the next hop. One can then repeat this analysis until the router which does not forward those packets is found, which would be the recipient.

**Tunnel Message** Message sent between two nodes that can contain one or more individual messages or parts of them.



# Bibliography

- [1] "Abscond browser bundle, official website (unavailable on 16.06.2016)." [Online]. Available: <https://hideme.today/>
- [2] "Bigbrother.i2p (i2p eepsite)." [Online]. Available: <http://bigbrother.i2p/>
- [3] "Bootstrap, official website." [Online]. Available: <https://getbootstrap.com/>
- [4] "codejava.net: Upload files to ftp server using urlconnection class." [Online]. Available: <http://www.codejava.net/java-se/networking/ftp/upload-files-to-ftp-server-using-urlconnection-class>
- [5] "Geolite2 free downloadable databases - maxmind developers site, official website." [Online]. Available: <https://dev.maxmind.com/geoip/geoip2/geolite2/>
- [6] "I2p documentation, garlic routing." [Online]. Available: <https://geti2p.net/en/docs/how/garlic-routing>
- [7] "I2p documentation, the network database." [Online]. Available: <https://geti2p.net/en/docs/how/network-database>
- [8] "I2p documentation, transport overview." [Online]. Available: <https://geti2p.net/en/docs/transport>
- [9] "I2p documentation, tunnel implementation." [Online]. Available: <https://geti2p.net/en/docs/tunnels/implementation>
- [10] "I2p documentation, tunnel message specification." [Online]. Available: <https://geti2p.net/en/docs/spec/tunnel-message>
- [11] "I2p observer, official website." [Online]. Available: <https://i2pobserver.security4web.ch/>
- [12] "I2p, official website." [Online]. Available: <https://www.geti2p.net>
- [13] "Installation guide for debian / ubuntu - i2p website, official website." [Online]. Available: <https://geti2p.net/en/download/debian/>
- [14] "Maxmind java api - maven central repository, official website." [Online]. Available: <https://search.maven.org/#search%7Cga%7C1%7Cg%3A%22com.maxmind.geoip2%22%20AND%20a%3A%22geoip2%22>
- [15] "Morris.js, official website at github.io." [Online]. Available: <https://morrisjs.github.io/morris.js/>
- [16] "Onion routing project, official website." [Online]. Available: <http://www.onion-router.net/>
- [17] "Privacy solutions, official website." [Online]. Available: <https://privacysolutions.no/>
- [18] "Raphaël javascrpt library, official website at github.io." [Online]. Available: <https://dmitrybaranovskiy.github.io/raphael/>
- [19] "Thehackerway, weblog (spanish)." [Online]. Available: <https://thehackerway.com/2011/12/19/preservando-el-anonimato-y-extendiendo-su-uso-hacking-i2p-escaneando-eepsites-y-descubrir-su-ubicacion-real-parte-x>
- [20] "The tin hat: I2p browser setup tutorial | using the tor browser for i2p (i2p eepsite)." [Online]. Available: <http://secure.thetinhathat.i2p/tutorials/darknets/i2p-browser-setup-guide.html>
- [21] C. Egger, J. Schlumberger, C. Kruegel, and G. Vigna, "Practical attacks against the i2p network," 2013.
- [22] E. Erdin, C. Zachor, and M. H. Gunes, "How to find hidden users: A survey of attacks on anonymity networks," *IEEE Communications Surveys & Tutorials*, vol. 17, issue 4, pp. 2296–2316, 2015.
- [23] M. Herrmann and C. Grothoff, "Privacy-implications of performance-based peer selection by onion-routers: A real-world case study using i2p," 2011.
- [24] J. P. Timpanaro, I. Chrisment, and O. Festor, "Monitoring the i2p network," 2011.

[25] "The Onion Router (TOR) project, official website." [Online]. Available: <https://www.torproject.org/>

# List of Figures

2.1. Screenshot of the I2P Router Console . . . . .	4
2.2. Creation of an Outbound Tunnel . . . . .	6
2.3. Creation of an Inbound Tunnel . . . . .	6
2.4. Basic structure of a Tunnel Message . . . . .	7
2.5. Multi-layered encryption of a single message during the individual stages of transport as seen by every participant . . . . .	8
2.6. Example for a Garlic Message between Alice and Bob. Source: <a href="https://geti2p.net/en/docs/how/garlic-routing">https://geti2p.net/en/docs/how/garlic-routing</a> . . . . .	10
3.1. Class Diagram of the I2P Observer . . . . .	15
3.2. Sequence Diagram of the I2P Observer . . . . .	16
3.3. Screenshot of the I2P Observer Website . . . . .	22
5.1. Overview of the Test Network . . . . .	26
6.1. Example of Clickjacking: A page with visible iframe (left) and how the user sees it (right) . . . . .	33
6.2. Warning displayed by I2P when trying to add a conflicting key to the address book . . . . .	33
6.3. Concept of Clickjacking Attack to add arbitrary entries to an address book . . . . .	34
6.4. Graph of reconstructed traffic path, reenacted with the High Traffic Attack . . . . .	36





# APPENDICES

## A. Configurations

### A.1. SSH Configuration

Example configuration of the ssh config file (`~/.ssh/config`) with the entries for one relay and the i2p-client:

```
Host bfh-BA-relay1
Hostname 10.0.0.11
User user
IdentityFile ~/.ssh/BFH_id_rsa
ProxyCommand ssh -W %h:%p bfh-BA-management
LocalForward 11211 127.0.0.1:7657
```

```
Host bfh-BA-client
Hostname 10.0.0.20
User user
IdentityFile ~/.ssh/BFH_id_rsa
ProxyCommand ssh -W %h:%p bfh-BA-management
LocalForward 11220 127.0.0.1:7657
LocalForward 14444 127.0.0.1:4444
LocalForward 14445 127.0.0.1:4445
```

### A.2. Shell script to control all relays

This script takes the command passed as parameter and executes it on every relay.

```
#!/bin/bash
COMMAND=${1}

ssh bfh-BA-relay01 $COMMAND
ssh bfh-BA-relay02 $COMMAND
ssh bfh-BA-relay03 $COMMAND
ssh bfh-BA-relay04 $COMMAND
ssh bfh-BA-relay05 $COMMAND
ssh bfh-BA-relay06 $COMMAND
ssh bfh-BA-relay07 $COMMAND
ssh bfh-BA-relay08 $COMMAND
ssh bfh-BA-relay09 $COMMAND
ssh bfh-BA-relay10 $COMMAND
```

## A.3. iptables

Output of iptables-save on the Management VM:

```
*filter
:INPUT DROP [0:0]
:FORWARD ACCEPT [31301:17151048]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -m state --state INVALID -j DROP
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 3 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 4 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 11 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 12 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A INPUT -m addrtype --dst-type BROADCAST -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -s 10.0.0.0/24 -d 10.0.0.0/24 -j ACCEPT
-A INPUT -s 10.0.0.0/24 -d 10.0.0.0/24 -p udp -j ACCEPT
-A INPUT -s 10.0.0.0/24 -d 10.0.0.0/24 -p tcp -j ACCEPT
-A INPUT -s 10.0.0.0/24 -d 123.123.123.0/24 -j ACCEPT
-A INPUT -s 123.123.123.0/24 -d 10.0.0.0/24 -j ACCEPT
-A INPUT -j LOG --log-prefix "[BLOCK INPUT] " --log-level 7
-A INPUT -j REJECT --reject-with icmp-port-unreachable
-A OUTPUT -o lo -j ACCEPT
-A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -p tcp -m state --state NEW -j ACCEPT
-A OUTPUT -p udp -m state --state NEW -j ACCEPT
COMMIT
# Completed on Wed May 11 11:30:54 2016
# Generated by iptables-save v1.4.21 on Wed May 11 11:30:54 2016
*nat
:PREROUTING ACCEPT [421:42293]
:INPUT ACCEPT [35:2645]
:OUTPUT ACCEPT [7:737]
:POSTROUTING ACCEPT [7:737]
-A PREROUTING -d 123.123.123.0/24 -j NETMAP --to 10.0.0.0/24
-A POSTROUTING -s 10.0.0.0/24 -j NETMAP --to 123.123.123.0/24
COMMIT
```

## B. Examples

### B.1. I2P Observer

#### B.1.1. Building I2P Observer

This section briefly describes how to build I2P Observer from its source code.

**Requirements:** I2P Observer relies on the package `net.i2p.data` from I2P, which unfortunately is spread across two branches: `core` and `router`. To retrieve both of them for the first build or to update them, follow the steps below.

- Fetch the source code of I2P from the Download site [12] or GitHub.
- Extract the archive
- Create a new project for the `core` branch, import the `/core/java/` folder from the source code and export the project as `.jar` file
- Do the same for the `router` branch in `/router/java/`
- Import both `.jar` files as libraries for the I2P Observer project

**Build I2P Observer:** I2P Observer has three dependencies: The two `.jar` files from the I2P source (build instructions above) as well as the MaxMind Java-API [14], including dependencies (Download: `geoip2-<version>-with-dependencies.zip`), need to be available as project libraries. The build process itself is simple.

- Adjust settings in `ObserverProperties` if needed
- Export I2P Observer as `.jar` file

#### B.1.2. Setup of I2P Observer

This instructions describe how I2P Observer can be set up on a new Linux host.

**Requirements:** To run I2P Observer on a host, the following requirements need to be fulfilled:

- Base system
- (suggested) own user for running I2P and I2P Observer
- I2P installed and configured to act as floodfill router
- Java 8
- Build of I2P Observer as described above

**Steps to set up I2P Observer:**

- Adjust username, I2P data folder, amount of runs per day and login credentials for uploading the statistic pages in the class `ObserverProperties` if needed (default: `username = user`, `I2P data folder = /home-/user/.i2p/`, credentials for `security4web.ch`) and rebuild I2P Observer
- (optional) Create a folder in which all data of I2P Observer is stored.
- Prepare the folder structure needed by I2P Observer: Create a folder `"html"` as well as a folder `"data"` which has the two subfolders `"daily"` and `"monthly"`
- Retrieve a copy of the GeoLite database [5] and store it as `geolite.mmdb` in the main folder

- Copy the .jar file of I2P Observer to the computer
- Create a cronjob to run I2P Observer at the desired intervals. For an hourly execution, which is the default, the cronjob entry could look like this: "00 \* \* \* \* java -jar /home/user/i2p-observer.jar > /dev/null"
- To verify that the settings are correct, a test run can be executed by hand ("java -jar i2p-observer.jar") which prints information if an error occurs

## B.2. Attacks on I2P

### B.2.1. Clickjacking Attack

Sample HTML code for a Clickjacking Attack. It only shows the minimal parts needed to embed the opaque iframe with the I2P Addresshelper page. The real destination of the I2P Eepsite was removed to make the code more readable.

```
<div style="position: absolute; left: 10px; top: 10px;">This is a example page
for clickjacking!</div>
<div style="position: absolute; left: 650px; top: 475px; color: red;
font-weight: bold;">Click here!</div>
<iframe style="opacity: 0;" height="550" width="800" scrolling="no"
src="http://malicious.i2p/?i2paddresshelper=<destination>"></iframe>
```