

# Anonymous Internet

Anonymizing peer-to-peer traffic using applied cryptography.

R.S. Plak

Delft University of Technology



# Anonymous Internet

Anonymizing peer-to-peer traffic using  
applied cryptography.

by

**Rutger Plak**

in partial fulfillment of the requirements for the degree of

**Master of Science**  
in Computer Science

at the Delft University of Technology,  
to be defended publicly on Monday July 7, 2014 at 1:30 PM.

Student number: 1358375  
Supervisor: Prof. dr. ir. J. A. Pouwelse

Thesis committee: Prof. dr. ir. H. J. Sips, Delft University of Technology  
Prof. dr. ir. J. A. Pouwelse, Delft University of Technology  
dr. ir. Z. Erkin Delft University of Technology

An electronic version of this thesis is available at  
<http://repository.tudelft.nl/>.



# Abstract

Throughout the years the Internet became of indispensable value to social, economic and political life. Despite this, the Internet in all its forms and protocols has proven to be extremely sensitive to manipulation by forces in control of the infrastructure of the Internet. Along with the growth of The Internet came the growth of censorship and filtering by governments. Governmental forces across the world have filtered, blocked and eavesdropped network traffic for both economic and political reasons.

This thesis focuses on the question of how filtering techniques and data censorship introduced by overseeing (governmental) forces can be circumvented. The challenge is to offer a fully autonomous and anonymized network of interconnected users with high-quality experience.

Along with this research, software has been implemented which enables the buildup of a file sharing network, where data is hard to trace and downloaders are indistinguishable from other users of the network. It forms the basis for a future anonymous network. This thesis and the developed software are a proof-of-concept which can and will be further developed by the Tribler team at the Delft University of Technology. This thesis introduces the 'p2p onion router', an anonymized network where high bandwidth can be achieved. This makes it more attractive than alternatives such as TOR for multimedia streaming. A new cryptographic approach based on the existing counter mode is introduced. This enables packet loss and packet reordering in stream ciphers, in order to utilize the benefits of BitTorrent traffic in onion routing.



# Contents

<b>List of Figures</b>	<b>vii</b>
<b>1 Introduction</b>	<b>3</b>
1.1 The illusion of privacy . . . . .	3
1.2 The goal of this work . . . . .	10
<b>2 Privacy enhancing technologies</b>	<b>11</b>
2.1 TOR . . . . .	11
2.2 OneSwarm . . . . .	13
2.3 Tarzan . . . . .	14
2.4 Flash proxy . . . . .	15
2.5 I2P . . . . .	16
<b>3 Problem description</b>	<b>19</b>
3.1 Cryptography requirements . . . . .	19
3.2 Usability requirements . . . . .	20
3.3 Attacks . . . . .	23
3.4 Adversaries . . . . .	25
<b>4 Design of p2p onion routing</b>	<b>29</b>
4.1 General outline . . . . .	29
4.2 Packet specification . . . . .	31
4.3 Packet encryption . . . . .	34
4.4 Block cipher mode of operation . . . . .	35
4.5 Opportunistic decryption . . . . .	37
<b>5 Implementation and experiments</b>	<b>43</b>
5.1 Synthetic experiments . . . . .	44
5.2 Profiling . . . . .	52
5.3 Tribler integration . . . . .	52
<b>6 Future work</b>	<b>57</b>
6.1 Anonymous seeding . . . . .	57
6.2 Resilience to attacks . . . . .	59
6.3 Circumventing infrastructure . . . . .	60
<b>7 Conclusion</b>	<b>63</b>
<b>Glossary</b>	<b>65</b>
<b>Acronyms</b>	<b>69</b>
<b>References</b>	<b>71</b>





# List of Figures

1.1	HTTP communication of Alice's request to Bob	4
1.2	Knowledge after HTTP request / response	4
1.3	HTTPS communication of Alice's request to Bob	5
1.4	Knowledge after HTTPS request / response	5
1.5	Schematic overview of BitTorrent traffic	7
1.6	Map of underwater cables	8
1.7	Sniffing fiber optic cables	9
1.8	Speed in TOR	10
2.1	Onion Routing	12
2.2	OneSwarm anonymity	13
2.3	Tarzan cover traffic	15
2.4	A flash proxy	15
2.5	Configuration of I2P	16
3.1	Illegal services in TOR	21
3.2	Child pornography in TOR	22
3.3	NSA's CottonMouth	25
3.4	NSA's PRISM and its competences	27
4.1	The CREATE packet	32
4.2	The CREATED packet	32
4.3	The candidate list	32
4.4	The EXTEND packet	33
4.5	The EXTENDED packet	33
4.6	The DATA packet	33
4.7	The PING packet	34
4.8	The PONG packet	34
4.9	Flow diagram for creating a circuit	35
4.10	Flow diagram for transmitting data through a circuit	35
4.11	Comparison of block cipher modes of operation	36
4.12	ECB encryption	36
4.13	CBC encryption	37
4.14	CTR encryption	37
4.15	Simplified diagrams of encryption of UDP packets	39
4.16	Simplified diagrams of decryption of UDP packets	40
4.17	Flow chart of the AES encryption per UDP packet	41
4.18	Flow chart of the AES decryption per UDP packet	41

5.1	The layered model for the p2p onion router . . . . .	43
5.2	Performance without cryptography . . . . .	44
5.3	Performance with cryptography . . . . .	45
5.4	The CPU usage with different circuit lengths . . . . .	46
5.5	Latency test with three circuit lengths . . . . .	47
5.6	CPU usage in opportunistic crypto . . . . .	48
5.7	Opportunistic cryptography performance with 0% packet loss . . . . .	49
5.8	Opportunistic cryptography performance with 0.1% packet loss . . . . .	50
5.9	Opportunistic cryptography performance with 1% packet loss . . . . .	50
5.10	Opportunistic cryptography performance with 5% packet loss . . . . .	50
5.11	Aggregated packet loss in onion routing (in percentage) . . . . .	51
5.12	Aggregated packet loss tested directly against LibTorrent . . . . .	51
5.13	Tribler's built-in profiler . . . . .	52
5.14	Anonymous downloading in Tribler . . . . .	53
5.16	The pull request . . . . .	53
5.17	Test download in Tribler . . . . .	54
5.15	Screenshot of Tribler running the p2p onion router . . . . .	55
5.18	Schematic view of anonymous download in Tribler . . . . .	56
6.1	The working of hidden services . . . . .	58
6.2	Anonymous uploading in Tribler . . . . .	59
6.3	Carrier NAT. Users of one provider share the same external IP. . . . .	60

# Preface

After the Snowden releases with information leaked from the NSA online privacy awareness has been a hot topic. It has been a shock to a lot of people that mass surveillance exists online even more than on the streets. But does the apparent existence of electronic mass surveillance harm the innocent? If innocent behaviour is to be defined as morally accepted behaviour then yes seems a plausible answer and Internet regulation seems for the greater good. But what if this morally accepted behaviour is defined by governments? And what if it is defined by governments whose ideology radically differs from common sense and human rights?

Internet monitoring and content filtering is an easy and effective way to regulate the Internet. Bypassing these monitors and content filtering techniques has proven to be a difficult task in which many have failed and none have truly succeeded. The complexity of the problem and the great amount of challenges that this task brings inspired me to do my thesis about *anonymous Internet*.

I would like to thank dr. ir. Johan Pouwelse for his continuous support and help. Furthermore I would like to thank Niels Zeilemaker whose experience with peer to peer networking and Tribler in particular has helped me enormously while implementing. I would also like to thank Zekeriya Erkin for his knowledge on cryptography. In particular I would like to thank Chris Tanaskoski with whom I've spend a lot of time with during the entire process and who has provided a base for implemented software. Last but not least I would like to thank my friends, family and my girlfriend for their support.

*Rutger Plak,  
Delft, June 2014*



# 1

## Introduction

In the developed world, over 77% of the population has access to the Internet. In the Netherlands and the United States of America this even reaches to 93% and 81% respectively<sup>1</sup>. People use the Internet as communication medium, news source, entertainment source and as business assistant. In daily social and economic life, the Internet plays an increasingly large role. The immense advantages that the Internet offers brings some obscure disadvantages that users are often not aware of. The recent Snowden leaks have given food for thought for a lot of Internet users. The NSA was long known to have collected records of over 1.9 trillion telephone calls with MAINWAY<sup>2</sup>, and Edward Snowden has revealed that this is nothing compared to the mass surveilling methods that are currently deployed. The NSA data center in Utah is even speculated to have yottabytes of storage<sup>3</sup>. This term is unfamiliar even for computer scientists, as it is so rarely used. One yottabyte equals  $10^{24}$  or 1.000.000.000.000.000.000.000 bytes. This is more than enough to have a high definition video and audio monitor of *every* person on the planet for 24/7 for *over a year*. The complete model, infrastructure and protocols used in the current Internet make mass surveillance possible on each layer of the system. This chapter will give an introduction on the problems in the current Internet.

### 1.1. The illusion of privacy

A visit to a website on the Internet exists of data going from the client to a server (the request), and data coming back from the server (the response). The most

---

<sup>1</sup>Source International Telecommunications Union: [http://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2013/Individuals\\_Internet\\_2000-2012.xls](http://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2013/Individuals_Internet_2000-2012.xls)

<sup>2</sup>More on MAINWAY here: [http://www.democracynow.org/2006/5/12/three\\_major\\_telecom\\_companies\\_help\\_us](http://www.democracynow.org/2006/5/12/three_major_telecom_companies_help_us)

<sup>3</sup>More information on the Utah data center: <http://siliconangle.com/blog/2013/07/29/the-mind-boggling-capacity-of-the-nsas-utah-facility/>

used form of the Internet is *HyperText Transfer Protocol*, abbreviated to *http*, standardised in 1996 [1]. A minimized and generalized explanation of the most used function of the http protocol is used to explain the basic working of the Internet.

## HTTP

Suppose *Alice* wants to make a connection to `http://tribler.org/index.html`, a packet of bits asking for `http://tribler.org/index.html` flows through cables to the server which we call *Bob*. Bob sends a response to Alice, consisting of bits in the form of packets. These bits represent the webpage, which Alice can interpret. Table 1.1 shows the simplified working of HTTP.

From	Packet	To
Alice	REQ, <code>http://tribler.org/index.html</code>	Server (Bob)
Server (Bob)	RES, <code>&lt;html&gt;&lt;head&gt;&lt;script src="scri</code>	Alice

Figure 1.1: HTTP communication of Alice's request to Bob

The infrastructure that is used in this communication, the cables, are most certainly not Alice's, and they're also not Bob's. Say *Chuck* is in control of the cables. Now Chuck can, if he wants, monitor, alter, filter or block the flow of data from Alice to Bob. Moreover, *Eve* may have been messing with Chuck's cables and may have placed a device on the cable that can monitor, alter, filter and block the flow of bits from Alice to Bob, without Chuck knowing.

The problem of HTTP is therefore that everybody that can reach the infrastructure can monitor, alter, filter and block bits flowing through the infrastructure. This type of attack is called the *Man in the middle* attack, in which a malicious node between two communicating nodes tries to get access to the data that is sent across the infrastructure. In order to circumvent this, HTTPS was introduced in 1994 and formalized in 2000[2]. Figure 1.2 shows who knows what after a HTTP request-response:

Entity	Knowledge
Alice	Alice requested a web page from Bob, Bob answered and gave the web page.
Bob	Alice requested a web page from Bob, Bob answered and gave the web page.
Eve	Alice requested a web page from Bob, Bob answered and gave the web page.

Figure 1.2: The knowledge of Alice, Bob and Eve after an HTTP request / response.

## HTTPS

*HyperText Transfer Protocol Secure* combines normal HTTP with secure *SSL certificates*. Instead of sending the bits directly over the cables, all bits are encrypted before they are sent. The bits are encrypted using the given SSL certificate, which is given by the webserver and validated by the client using *trusted entities*. The request that Alice makes does not consist of "Give me `http://tribler.org/index.html` please" anymore, but exists of the hostname and encrypted data, which looks random to anybody without the certificate. Bob does have the certificate and can

*decrypt* the request. Bob encrypts the response with the certificate of Alice, so nobody except Alice can read the response. Figure 1.3 shows the simplified working of HTTPS.

From	Packet	To
Alice	tribler.org098f6bcd4621d373cade4e832627b4f6b	Server (Bob)
Server (Bob)	fa350a2ad4cb6b89db3230b7a04c77ec13bd1146bc8a	Alice

Figure 1.3: HTTPS communication of Alice's request to Bob

This solution makes sure that a man in the middle can no longer monitor the data. He can no longer read what Alice and Bob are doing, and can therefore not easily filter the data based on content. Still however, a man in the middle knows that Alice and Bob are communicating. Therefore he can still block content and alter content (making it undecryptable and thus invalid).

The design of HTTPS lets clients trust a number of certificate authorities for validating the certificates that are given by webservers. This directly forms a problem as these certificate authorities are not immune to hacks and can not be trusted with absolute certainty. A famous example of such case is Diginotar, a Dutch certificate authority that was hacked and issued malicious certificates for Google, Facebook and many more<sup>4</sup>.

Furthermore, malicious entities can still log all encrypted data. When at a certain point in the future Alice is hacked and her private key is compromised, the malicious entity can decrypt all previous traffic between Alice and Bob. There is no built-in form of *Perfect Forward Secrecy*, in which certificate theft does not give away previous session keys. Figure 1.4 shows who knows what after a HTTPS request-response:

Entity	Knowledge
Alice	Alice requested a web page from Bob, Bob answered and gave the web page.
Bob	Alice requested a web page from Bob, Bob answered and gave the web page.
Eve	Alice and Bob have communicated and I know the encrypted data.

Figure 1.4: The knowledge of Alice, Bob and Eve after an HTTPS request / response.

Both HTTP and HTTPS lack anonymity and are easy victims for a mass surveilling entity. HTTPS adds certificates to hide the transferred data, introducing end-to-end cryptography. This end-to-end cryptography relies on trust in several<sup>5</sup> root certificates signed by companies of which over 50% are from the United States of America<sup>6</sup>. One could argue that this trust is not based on anything. The re-

<sup>4</sup>More on Diginotar in <http://www.rijksoverheid.nl/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update.html>

<sup>5</sup>Both a clean Firefox installation and a clean Google Chrome installation trust all certificates issued by 94 authorities by default. For Firefox: Settings → Advanced → Certificates → View Certificates → Authorities. For Chrome: Settings → Manage Certificates → Authorities.

<sup>6</sup>Certificate statistics: [http://w3techs.com/technologies/overview/ssl\\_certificate/all](http://w3techs.com/technologies/overview/ssl_certificate/all)

cent appearance of the Heartbleed bug introduces a fatal flaw in an often used implementation of the HTTPS trust model.

## Heartbleed

Due to the bug 'Heartbleed', HTTPS was *not* secure for a very long time. The complete foundation on which HTTPS is built was compromised, making the S for secure in HTTPS vanish in thin air. HTTPS relies on SSL certificates. These SSL certificates can and are mostly handled by a framework called *OpenSSL*. This framework has proven to be vulnerable to a buffer-over-read attack, in which 64 KB of active server memory can be confiscated per attack. The attack leaves no trace and can be done as often as needed in order to read the entire memory. This means that all active certificates, keys, passwords etc. are insecure<sup>7</sup>. Experts call the bug 'catastrophic'<sup>8</sup>, in which over 600.000 websites are vulnerable, including Wikipedia, Twitter, Yahoo, Dropbox and many more. As of the day of this writing, there are still over 300.000 unpatched websites<sup>9</sup>. The underlying technology for HTTPS is easily compromised with the bug. This completely defaces the power of HTTPS and even worse, it gave users trust where no trust was due.

Not only the technical aspects of HTTPS are broken with Heartbleed. There are even speculations that the National Security Agency knew about the Heartbleed bug for over two years<sup>10</sup> before the bug was announced in April 1st 2014. It is clear that even HTTPS encrypted traffic can be monitor, altered, filtered and blocked based on content by governmental forces such as the NSA, as all private keys can be read from server memory. There is and has been no privacy with both HTTP and HTTPS web browsing.

## BitTorrent

BitTorrent is an approach to the Internet which removes the servers. A so called *torrent* file holds information about the download of single file or multiple files. The torrent file also contains information about *trackers*, which keep track (hence the name) of who is downloading the file. Instead of downloading information from a server, the information is downloaded from other users of the same torrent file. When you are connected to ten people which have parts of the file, you can download the parts that they already have in parallel. In return you give them parts that they don't have and you do. This mechanism is however not anonymous at all. It is even extremely easy to identify users that are downloading a certain file, just by asking the tracker. Figure 1.5 gives a schematic overview of BitTorrent network connectivity.

<sup>7</sup>A simple but very accurate explanation about Heartbleed here <http://xkcd.com/1354/>

<sup>8</sup>Bruce Schneier, <https://www.schneier.com/blog/archives/2014/04/heartbleed.html>

<sup>9</sup>Security scan: <http://blog.erratasec.com/2014/06/300k-vulnerable-to-heartbleed-two.html>

<sup>10</sup>Bloomberg article: <http://www.bloomberg.com/news/2014-04-11/nsa-said-to-have-used-heartbleed-bug-exposing-consumers.html>



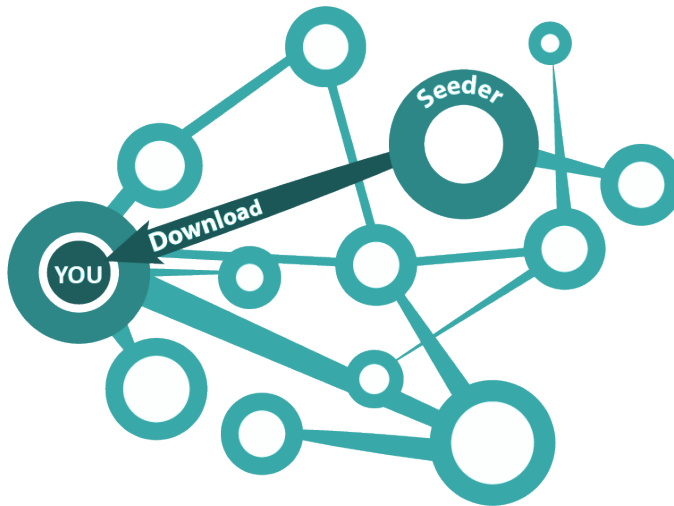


Figure 1.5: Schematic overview of BitTorrent traffic. Every node in the swarm shares its content of the torrent file. The *you* node is downloading the file by asking other people in the BitTorrent swarm for peaces of the file in parallel. This increases download speed but downloaders of the files are easily traceable. (Image by Myra Vreede.)

## Compromising the infrastructure

It is clear that once the infrastructure is compromised by an adversary, the complete internet traffic is compromised even if it's encrypted. And the infrastructure is indeed compromised. Internet traffic flows through hundreds of thousands of miles of underwater cables (see figure 1.6), connecting the world. These cables handle all (encrypted) Internet traffic and are therefore of extreme importance to an entity that wants to monitor the system. Glimmerglass is a government contractor of the United States of America which develops wiretapping equipment that can tap all information that flow through the cables, independent of the data rate, protocol and format (see figure 1.7).

The tools developed by Glimmerglass still need to be attached to the under-sea cables. During the cold war, the NSA tapped undersea cables that handled all communication traffic between Soviet army bases in Operation Ivy Bells. The sea where the cables were placed were a no-go zone which was protected by the Soviet army. The NSA solved this by placing undersea wiretapping devices using submarines. The same trick might just be repeated nearly 50 years later, with the submarine USS Jimmy Carter. This submarine is capable of wiretapping undersea cables with sophisticated devices (such as the ones developed by Glimmerglass). It isn't very likely that the USS Jimmy Carter is monitoring the Facebook status of a random citizen anywhere in the world, but if the NSA wanted to, they could<sup>11</sup>.

<sup>11</sup>More on the USS Jimmy Carter here <http://blogs.reuters.com/great-debate/2013/07/18/the-navys-underwater-eavesdropper>. More on Operation Ivy Bells here [http://www.military.com/Content/MoreContent1/?file=cw\\_f\\_ivybell](http://www.military.com/Content/MoreContent1/?file=cw_f_ivybell)

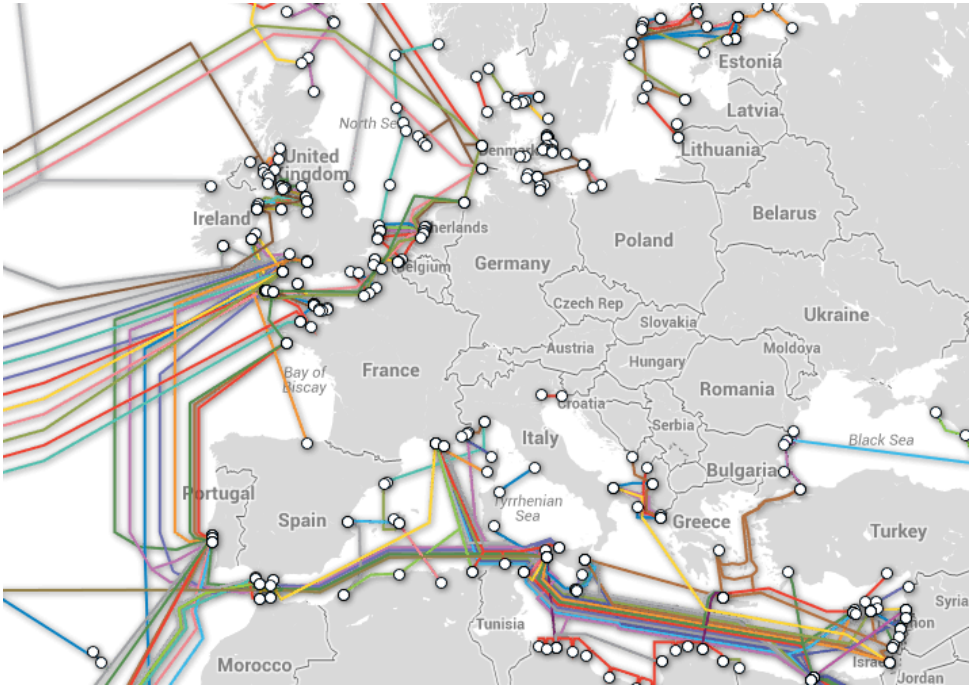


Figure 1.6: Underwater cables in Europe. There are hundreds of thousands of miles of cables throughout the oceans in the world. These cables together connect the entire world in the Internet. An interactive version of this map is available here <http://www.submarinecablemap.com/>

## TOR

So far there have been several attempts to circumvent governmental monitoring. TOR is the most widely used anonymous Internet. With over 1.000.000 daily users<sup>12</sup> no other system comes even close to this usage level. *The Onion Router*[3] tries to circumvent mass surveillance using 'onion routing'. The Onion Routing paradigm ensures that no entity knows who is communicating with who, nobody can intercept sent data and nobody can alter the data without anybody noticing it. The concept of onion routing is straight forward and relies on asymmetric cryptography. Suppose  $A, B, C$  and  $D$  are mathematical functions which are reversible by  $A^{-1}, B^{-1}, C^{-1}$  and  $D^{-1}$  respectively. Alice created functions  $A$  and  $A^{-1}$  and gives  $A$  away whilst keeping  $A^{-1}$  private. Anybody that wants to send a message to Alice sends  $A(\textit{hello})$  instead of *hello*. This way only Alice can decrypt the message. Instead of sending a request directly through a server, the request travels multiple hops. Alice sends  $B(C(D(\textit{hello})))$  to Bob, which peels his 'onion layer' with  $B^{-1}$  and forwards  $C(D(\textit{hello}))$  to Charlie. Charlie peels his onion layer and forwards it to Dave. Now Dave has no idea about who did the request, and all other nodes don't know the content of the request. This paradigm has been utilized in multiple anonymous

<sup>12</sup>See <https://metrics.torproject.org/users.html> for accurate user statistics. Peaks of this month at about 2.000.000 users.

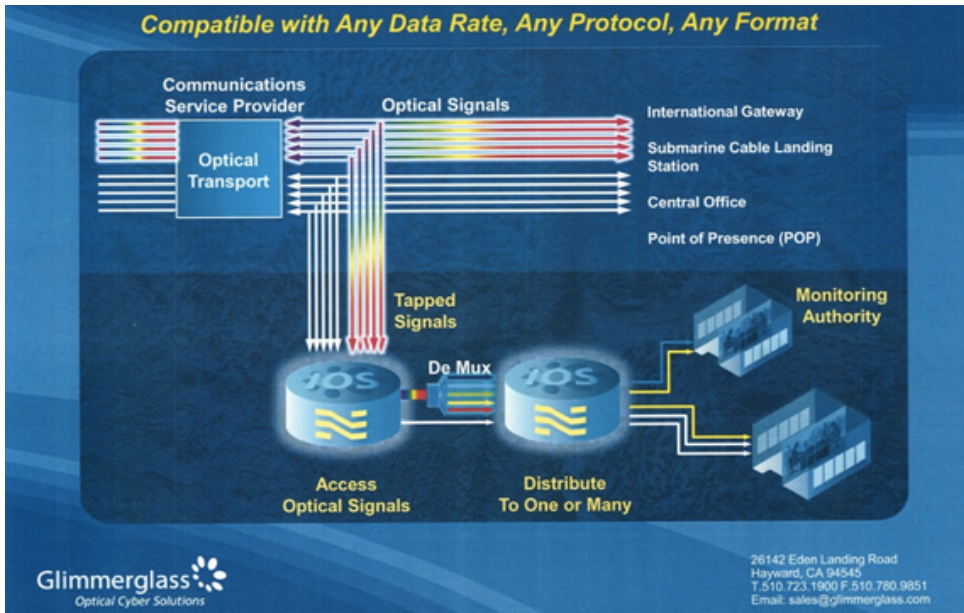


Figure 1.7: Glimmerglass develops tools which can tap all fiber optic cables. This is a slide from the presentation of Glimmerglass. Source : "The Atlantic",

<http://www.theatlantic.com/international/archive/2013/07/the-creepy-long-standing-practice-of-undersea-cable-tapping/277855/>

systems, including the design and software developed in this thesis. TOR offers anonymous downloading and anonymous content sharing. The problem with TOR is that it is very slow. TOR allows between five and ten seconds per megabyte (figure 1.8), which translates to 100-200 KBps. This is by far not fast enough for an acceptable user experience. More on onion routing and TOR in section 2.1.

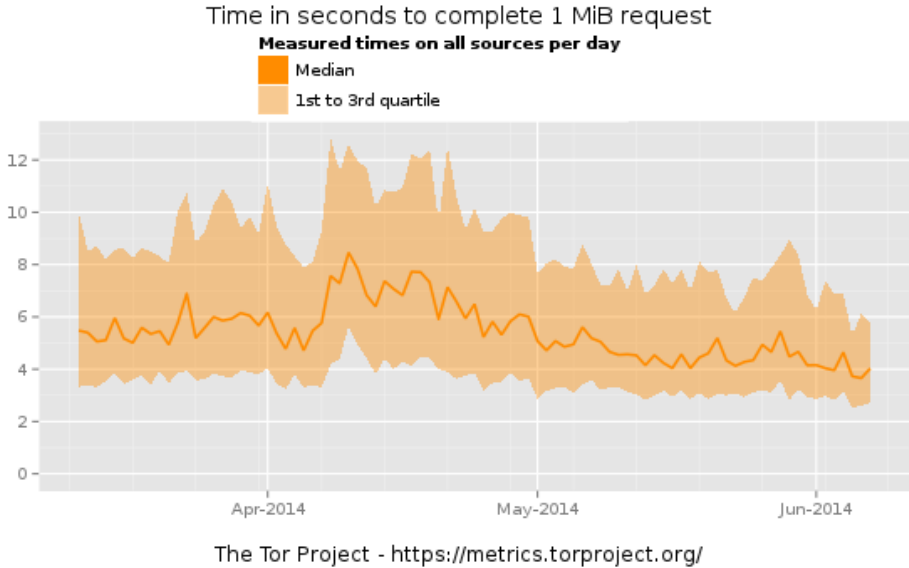


Figure 1.8: Speed in TOR. The graph shows that a 1MB downloads takes more than 5 seconds.  
(Source: <https://metrics.torproject.org/performance.html>)

## 1.2. The goal of this work

The problem with the current Internet is that it is not anonymous nor does it protect privacy. One of the major advantages of the current Internet is the high bandwidth that can be utilized. This is the number one problem with the privacy enhancing technology TOR, it is unacceptably slow. The 100-200 Kbps is far too slow for any form of multimedia whatsoever. The goal of this work is to introduce the first step in a new *anonymous Internet*, which respects the privacy of the user and unlike TOR utilizes as much bandwidth as needed for a good user experience. In the current multimedial era this means that it should be easily possible to stream a High Definition movie of 1280x720 pixels with high definition audio, translating in roughly 10Mb/s (1.25MB/s). The research done in this thesis is deemed to provide the first step in a high bandwidth, anonymous Internet. Chapter 2 will cover past attempts in creating an anonymized Internet. Chapter 3 will state the problems that arise when developing such a system. An approach of deploying onion routing in peer-to-peer (p2p) traffic is introduced in chapter 4. The implemented version of this protocol is tested in chapter 5. Chapter 6 will cover future work and this thesis concludes in chapter 7.

# 2

## Privacy enhancing technologies

Recent research has shown that 86% of the people have taken some sort of measures to erase their digital footprint<sup>1</sup>. 68% of the people think current laws don't enforce online privacy enough and 50% of the people is concerned about the amount of data that is available online about them. The lack of understanding of the underlying technologies makes it hard for people to understand to what extent privacy can and will be preserved. Furthermore non-transparency of large companies contribute to the fear of Internet users to lose their privacy completely. This makes the Internet a more and more conservative medium, in which there is feared to be no privacy, let alone anonymity, whatsoever.

Over the years several attempts have been made to enhance privacy within the Internet. These attempts all either lack usability, or their technical design makes them unfeasible for real world usage. This chapter will cover the five attempts which have or had the highest potential of breaking through.

### 2.1. TOR

TOR (The Onion Router) is a protocol that allows anonymous connections to be setup and used. It was developed in 2002 and has been under development since. The goal of TOR is to offer an anonymous platform in which users are untrackable and censorship is impossible. The system relies on the voluntary bandwidth donation of users. Content is not directly fetched after a request, but circuits consisting of multiple users are built over which data is transferred. Leaked NSA files describe TOR as *"The king of high secure, low latency Internet Anonymity"*, with

---

<sup>1</sup>Research: <http://www.pewinternet.org/2013/09/05/anonymity-privacy-and-security-online/>

“No contenders in the waiting”<sup>2</sup>. The cryptographic design of TOR makes it (nearly) impossible to identify users and filter content. Hidden services allow untrackable websites to exist and offer content without being identifiable or filterable.

## 2

### Working of TOR

TOR nodes create circuits of multiple hops in a fashion that each hop only knows its neighbors and no more, and the originator has shared a secret with each hop. When the originator wants to do a data request it wraps the request in layers using the shared secrets that it has with the hops. These are the *Onion layers*, hence the name Onion Routing. Each hop peels its layer and forwards the message. The exit node can read the message, does the data request and sends the data back with its own layer around it. Each hop adds its own layer and the originator of the request can peel all layers as he has the shared blue secrets<sup>3</sup>.

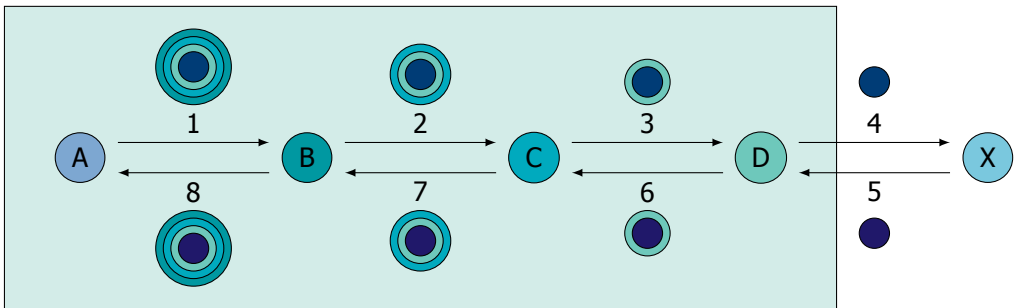


Figure 2.1: Onion Routing

### Cryptography in TOR

TOR uses a RSA 1024 PKI with OAEP-MGF1 padding and SHA-1 as digest[4–6]. Furthermore it uses the ed25519 elliptic curve for handshakes[7]. Data streams are encrypted with AES 128 bit in counter mode with initialisation vector of pure zeros[8]. Shared key generation is done with Diffie Hellman and the 1024 bit prime defined in RFC2409[9, 10].

### Problems of TOR

TOR only handles TCP streams. It is therefore unusable for any application not using TCP. Furthermore TOR lacks speed. TOR only offers between 100 and 200 KBps, which is insufficient for even YouTube streaming. There are several design issues that lead to this disadvantage. See [11] for more information.

1. There are a limited number of exit nodes, which handle *all* TOR traffic.

<sup>2</sup>Source: <http://www.theguardian.com/world/interactive/2013/oct/04/tor-high-secure-internet-anonymity>

<sup>3</sup>More crypto at [https://gitweb.torproject.org/torspec.git;a=blob\\_plain;hb=HEAD;f=tor-spec.txt](https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=tor-spec.txt)

2. There are a limited number of entry nodes, which handle *all* TOR traffic.
3. TCP congestion control in TOR overloads circuits. When two nodes share multiple circuits, all circuits take the same TCP stream. This leads to the advantage that the circuits are indistinguishable by an observer, but when one of the circuits sends too many bytes the congestion control slows down the entire TCP connection. This causes all other circuits between the nodes to slow down, even though this is not necessary.
4. There is unequal load balancing. A small number of users use a large amount of data. This slows down the network for the users that use less bandwidth. There is no bandwidth donation or tit-for-tat implementation in TOR.

## 2.2. OneSwarm

OneSwarm is a file sharing tool which enables sharing files with friends only. OneSwarm has a small but dedicated user community in Sweden and France as indicated by their support forum<sup>4</sup>. It is an open source software package that aims to make p2p traffic anonymous. It is developed at the University of Washington to be *more anonymous than BitTorrent* and at the same time to be *faster than TOR*. OneSwarm allows user to share data with certain people whilst remaining anonymous for other people. The social aspect in OneSwarm makes it possible for both the sender and the receiver of data to remain anonymous or not. When anonymous, messages and requests are forwarded with a certain probability, and a response will be sent only after a delay, see figure 2.2. This way when somebody tries to query you or fetch data from a node, that node has a plausible deniability that the answer of the request wasn't actually his.

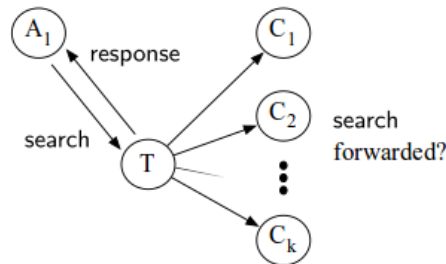


Figure 2.2: The basic idea of OneSwarm's anonymity model. A search query is forwarded with a certain probability. Node A can't tell whether the answer comes from T or from another node. Images source: original paper[12]

### Cryptography in OneSwarm

OneSwarm is backwards compatible with BitTorrent and data can be transferred in plain sight. When users share files privately, OneSwarm uses RSA for asymmetric

<sup>4</sup>See the forums at <http://forum.oneswarm.org/>

encryption and AES 256 bit for symmetric encryption[4, 8]. Hashing is done using SHA-1 digests[6].

### Problems of OneSwarm

OneSwarm has been shown [13] to be vulnerable for a lot of attacks just by looking at the plain data. Furthermore, when searching unpopular items on OneSwarm, the search propagates through a lot of nodes. Because nobody can return results for the query, it has to be sent to a large number of nodes. This increases the traffic that is caused by each search query. Downloads are faster when the number of hops are smaller. Therefore when a OneSwarm node tries to optimize its performance it will become friends with more people. This directly counters the social aspect as trust is off the borders.

### 2.3. Tarzan

Tarzan[14] is an abandoned research prototype of a peer-to-peer system developed by the New York University and Massachusetts Institute of Technology in 2002. It has zero usage. It is built as an application independent IP system and uses layered cryptography just like TOR. Every user must know the entire system and all have to be connectible. Initiators choose their paths by themselves using pseudo-random node selection. Tarzan uses cover traffic against eavesdroppers, and only uses paths for 'real' data transmission when cover traffic also flows across the path. The last node in the path does the request to the outside world using his Network Address Translation and relays the data back to the originator.

#### Cryptography in Tarzan

Packets are encrypted using Blowfish in CBC mode with an 8 byte random initialization vector per packet[15]. Sha-1 is used for hashing and Rabin is used for asymmetric cryptography[6, 16].

#### Problems in Tarzan

Every node in the system has to know *all* other nodes in the system. This implies that when the userbase grows the knowledge per node grows equally. This makes Tarzan inscaleable when it comes to millions of users. On top of that, the network is enduring excessive load due to the large amounts of cover traffic that flow through the system. This is undesirable, especially when users use mobile devices such as smartphones or tablets which have limited data capacity. Figure 2.3 shows the cover traffic of Tarzan. The software built along with the research is extremely outdated and isn't maintained.



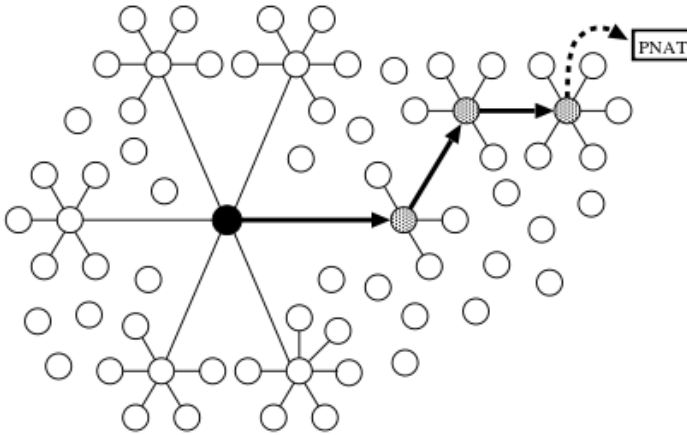


Figure 2.3: The cover traffic in Tarzan. The 'useful' data is shown in thick black lines. The thin lines are cover traffic, which increases the load of the network. Image source original paper[14]

## 2.4. Flash proxy

Flash proxy[17] is a prototype which expands TOR by adding entry nodes, developed at Stanford University in Stanford. As with Tarzan it has very little to no usage<sup>5</sup>. It is created to circumvent filtering mechanisms in countries where governmental forces block content on the Internet. In TOR (see section 2.1) a limited number of entry nodes is available. They are easily blocked by governmental forces, making TOR unusable. A *flash proxy* is a proxy that operates as a small badge on a web page, see figure 2.4. When somebody visits the page he is automatically an entry node for TOR, allowing other users to enter TOR using the badge as entrance. The client that runs the badge forwards all the traffic for them. The fact that the IP addresses of the people that visit the webpage is dynamic, governmental forces cannot block them easily.



Figure 2.4: The badge as shown on a webpage when the proxy is active. Dark blue is idle, light blue is active (somebody is using your proxy). Source of the badge:

<http://crypto.stanford.edu/flashproxy/>

### Problems of Flash proxy

The person that is running the flash proxy (the badge) has to be in an area where TOR is reachable. If the server of the web page is in an area where TOR entry

<sup>5</sup>See up-to-date measurements at <https://metrics.torproject.org/users.html?graph=userstats-bridge-transport&transport=websocket#userstats-bridge-transport>

nodes are also blocked, there is still no entry in TOR. Furthermore flash proxy does not hide the person using the badge as an entry for tor. Any entity can start a browser badge and see that person  $x$  is connecting to him, wanting to make use of TOR. A big design issue is that the default browser badge is implemented as an iFrame that links to <http://crypto.stanford.edu/flashproxy/embed.html>. Blocking \*.stanford.edu is enough for a government to shutdown flash proxy. Even if the browser badge is embedded from another domain, it is still easy to block all websites that contain browser badges.

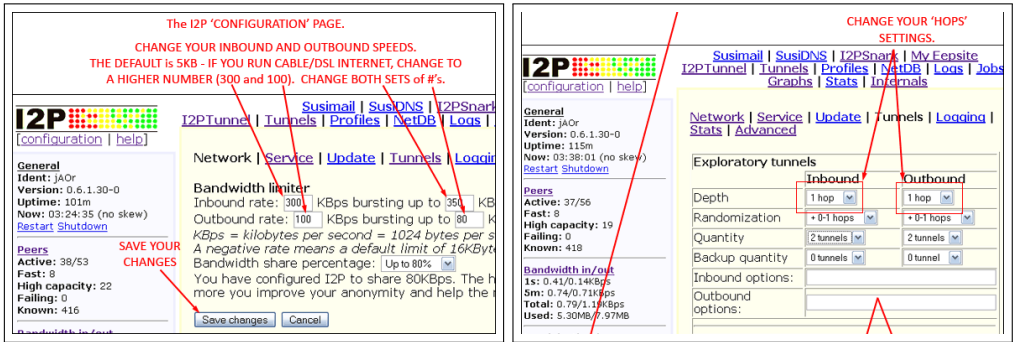


Figure 2.5: Configuration of I2P. Images source:

<http://filesharefreak.com/2007/12/16/darknets-private-internet-file-sharing>

## 2.5. I2P

I2P, short for Invisible Internet Protocol is a network architecture that provides anonymous and secure network traffic. When two nodes are communicating, they send their traffic over their outbound tunnels towards inbound tunnels of the recipient. The origin and destination are never more than an anonymous cryptographic key. I2P can be used as a network layer for existing applications such as BitTorrent, IRC and email. It includes websites accessible only through I2P, so called *eepsites*. These eepsites are servers setup locally with inbound tunnels pointing towards them.

### Cryptography in I2P

I2P uses Elgamal as asymmetric encryption method and AES 256 bit in CBC mode for symmetric message encryption[8, 18]. Session keys are negotiated using 2048 bit Diffie Hellman[9]. 1024 bit DSA is used for signatures and SHA256 is used as

hashing function for generating encryption keys[19, 20].

### Problems in I2P

I2P tries to be a replacement for the IP protocol. This immense goal introduces a configurational nightmare. It is near impossible to configure I2P for people not completely familiar with terminology that only people with a masters degree in computer science understand. This complete lack of usability constructs a big gap between the great potential of I2P and the small userbase. See figure 2.5 for a small portion of configuration settings in I2P.



# 3

## Problem description

The Internet is under attack. In its current form the Internet does not comply to any privacy enhancing standards and it does not offer anonymity. In order to increase privacy and anonymity in an online setting, the ability of organisations to *eavesdrop traffic*, *alter traffic* and *filter content* must be reduced to a minimum and eliminated completely if possible. The Internet Engineering Task Force has recently published rfc7258[21], in which their concern about mass surveillance is discussed. They state that *"Pervasive monitoring is a technical attack that should be mitigated in the design of IETF protocols, where possible"*. The goal of this thesis is to introduce the first step of a content sharing tool alternative to the 'Open' Internet, that utilises the existing infrastructures of the Internet, but is immune to organisational surveillance.

The problem that is inherit to data sharing and the transportation of bits is the vulnerability of the data. All bits that move from A to B can be monitored, altered, filtered or communications can be blocked completely. This chapter covers the problem that have to be solved and the problems that appear when introducing solutions for the former problems.

### 3.1. Cryptography requirements

The main privacy problems that are pointed out in the Introduction that need to be solved using cryptography in an anonymous Internet are listed and described in arbitrary order.

**Monitoring** A malicious entity should *not* be able to obtain any information about the content that is transfered between two arbitrary entities.

**Altering** A malicious entity should *not* be able to alter any information that is transfered between two arbitrary entities without them noticing it.

**Filtering** A malicious entity should *not* be able to filter any transferred information based on its content.

**Traceability** A malicious entity should *not* be able to obtain sustainable information to prove that data is being sent from entity A to entity B.

**Perfect Forward Secrecy** Once an entity is compromised and its private key is compromised, any data that has been logged overtime should *not* be decryptable.

## 3

### 3.2. Usability requirements

Often privacy enhancing technologies fail to become popular outside the core user community that compiles their own kernel and that uses PGP<sup>1</sup>. Seamless operation and auto configuration is essential for large-scale uptake. An anonymous Internet needs not only to comply to the cryptographic requirements, the system should focus on usability. TOR, OneSwarm, I2P and Tarzan all comply to (most of) the cryptographic requirements. TOR however lacks speed, OneSwarm, I2P and Tarzan are a nightmare to understand and configure. We address the usability requirements of an anonymous Internet and discuss the problems that are to be encountered in alphabetical ordering.

#### Bandwidth utilization and network load

The major disadvantage of TOR is its speed. Speeds of about 200 KBps (see figure 1.8) are insufficient for a user friendly anonymous Internet. When dealing with any kind of onion routing, nodes have to relay traffic and for every byte that is requested, this byte also travels through the proxies. This means that users will have to donate bandwidth in order to let other people download. In BitTorrent this is fixed using the 'tit-for-tat' mechanism, which prevents users from *leeching* by giving higher bandwidth to users that upload a lot[22]. An incentive system with voluntary relaying / uploading should be introduced to prevent a lack of network capacity.

#### Distributed nature

In order to avoid a single point of failure or a single point of trust, an anonymous Internet should be decentralized. This removes the ability of governmental forces to disable the single point of failure (or multiple, if applicable). Sometimes this is nearly impossible. During the Egyptian revolution, the Internet was completely shut down by the government simply by eliminating the Internet service providers<sup>2</sup>. Such a shutdown is impossible to prevent. For the sake of security, web browsers trust multiple root certificates by default, see section 1.1. When an issuer of such root certificate is compromised, the entire system is not to be trusted. Single points of failure and trust should be avoided where possible.

<sup>1</sup>Lots of real technical questions on the TOR, OneSwarm and I2P forums. More on <http://tor.stackexchange.com/>, <http://forum.oneswarm.org> and the eepsite <http://forum.i2p/> (only accessible through I2P)

<sup>2</sup>"Live" feed of the disconnection of the Internet: [http://www.huffingtonpost.com/2011/01/27/egypt-internet-goes-down-\\_n\\_815156.html](http://www.huffingtonpost.com/2011/01/27/egypt-internet-goes-down-_n_815156.html)

## Migration path

It is infeasible to solve all problems at once with one system. Quality mechanisms and concepts should be utilized and where possible the software should be kept backwards compatible. Keeping backwards compatibility with BitTorrent will for example give the free advantage of available content. The modularity of a network protocol for anonymous Internet can be of great value to other applications.

## Moral high ground

Unfortunately privacy enhancing technologies are not only used as a force of good. One of the main purposes of monitoring, content filtering, content blocking and the mass surveillance by governmental forces is 'catching criminals'. Terrorist networks, drugs and armory distributing networks such as Silk Road [23] and child pornography networks are a disgrace to the Internet and are in no way desirable. Still mass surveillance on a person should arguably be the exception instead of the rule. Furthermore, 'criminals' is defined by the governmental forces itself. What if having a sexual orientation that is illegal in a country? Should governments be able to log all traffic in order to find out a persons sexual orientation? The extend to with the mass surveillance is possible in the Internet in its current form raises a lot of these questions.

### [ [EDIT](#) ] Commercial Services

Buying and selling stuff, auction houses, sales forums, gaming.

See also: *The seperate Drugs and Erotica sections for those specific services.*

- [Contract Killer](#) - Kill your problem (snitch, paparazzo, rich husband, cop, judge, competition, etc).
- [Bit Poker v1.93](#) - Poker (Bitcoin).
- [Buttery Bootlegging](#) - Get any expensive item from major stores for a fraction of the price!
- [Stat ID's](#) - Selling fake ID's.
- [Bitcoin](#) - Like Ebay. We increase the gross national product.
- [Video Poker](#) - A casino that features 'Jacks or better' video poker.
- [Cheap SWATTING Service](#) - Calls in raids as pranks.
- [Data-Bay](#) - Buy and sell files using digital currency.
- [The Last Box](#) - Assassination Market (Bitcoin).
- [Pirax Web DDos](#) - Take out your enemies in seconds.
- [Hacking Services](#) - Hacks IM and Social Nets, does DDos, sells bank/credit/paypal accounts. Se habla Espanol.
- [Email Hacker](#) - Hack emails (Bitcoin).
- [CC4ALL](#) - Selling valid Credit-Cards. Most from Germany.

Figure 3.1: Services offered on TOR, page easily found using the 'Hidden Wiki'. (Source: <http://krypt3ia.wordpress.com/2011/09/04/the-hidden-wiki-between-the-layers-of-the-onion-router-networks/>)

In constructing a system where mass surveillance is impossible there are a lot of moral issues that have to be addressed. How are these 'criminals' defined? How do you prevent assassins, terrorists and pedophiles from misusing such a network? The characteristics of such system are theoretically an ideal base for such malicious

users to switch to such a system. TOR for example is misused by cyber criminals in many ways<sup>3</sup>. Figure 3.1 shows just a grasp of services offered on TOR and show the cruel activities that are offered on TOR.

Almost all of these offered services are illegal. One could argue that these services are offered and used anyway and TOR is just another medium for these services. But what about child pornography? Figure 3.2 shows how easy it is to find child pornography on TOR.

Some internal organ of an anonymous Internet should be responsible for recognizing such content and its users. This however raises a censoring aspect to the system, which is exactly one of the problems in the current Internet, as censoring aspects of such a network can easily be abused by governmental forces. One could also argue that even if the criminals use the system and are theoretically untrackable, they will reveal themselves by human error. An officer tracing investigating child pornography in TOR says that they have been able to arrest 25 people by looking at traces they leave: *"There's not a magic way to trace people [through Tor], so we typically capitalize on human error, looking for whatever clues people leave in their wake," ... "that has so far resulted in 25 arrests and the identification of more than 250 victims, all children."*<sup>4</sup>.

#### PROJECTS

- [History of CP](#) - An attempt to make an encyclopedia of Child Pornography, Child Models, MySpace girls and the like.
- [Centralization Project](#) - For now it is just discussion about creating a distributed wiki-like database complete with image samples and file hashes.
- [To do](#) - List of suggestions to reconcile the society with child lovers.
- [Your Own Pedo Site](#) - A tutorial on making a pedo site of your own, and reducing the attendant anonymity/security risks.
- [Pedo tag](#) - Discussion on how to identify the pedo-community.
- [Back to Normal](#) - Blueprint for a new child love movement.
- [Pedophilen-Gemeinschafts-Bund](#) - German interest and self help group.
- [The Upload Guide](#) - Project to setup a guide for cleanet uploads, includes host list.
- [The Child Nudity Wiki](#) - An indexing of all instances of child nudity.

Figure 3.2: Screenshot of TOR hidden wiki, the ease of finding child pornography. (Source: <http://krypt3ia.wordpress.com/2011/09/04/the-hidden-wiki-between-the-layers-of-the-onion-router-networks/>)

## NAT compatibility

Users that use an anonymous Internet should be able to make use of a firewall or Network Address Translator (NAT) without having to make themselves connectible. This enables the disadvantage that ports can be closed and people cannot connect

<sup>3</sup>A lot of examples here: [http://www.securelist.com/en/blog/8187/Tor\\_hidden\\_services\\_a\\_safe\\_haven\\_for\\_cybercriminals](http://www.securelist.com/en/blog/8187/Tor_hidden_services_a_safe_haven_for_cybercriminals)

<sup>4</sup>Wall Street Journal article: [http://online.wsj.com/news/articles/SB10001424052702303949704579461641349857358?mod=WSJ\\_TechWSJD\\_NeedToKnow](http://online.wsj.com/news/articles/SB10001424052702303949704579461641349857358?mod=WSJ_TechWSJD_NeedToKnow)



to each other. When A can communicate with both B and C, this does not necessarily mean that B and C can communicate. This can be solved using NAT puncturing[24], but this could reveal identities. An anonymous Internet should either respect the NAT problem and circumvent it or it should apply NAT puncturing in a fashion that does not reveal identities.

### Open Source

Making an anonymous Internet open source will bring a lot of free advantages. Developers across the world will have access to the code and will be able to alter the code. This brings the advantage that bugs will be found faster, they will be fixed faster, users will be able to increase the performance and add their knowledge to the software. Furthermore it is much easier to trust transparent software. This way the users can see the complete working of the software themselves, increasing trust in the software.

### Scalability

An anonymous Internet should perform better or just as good when the number of users grows. The system should be designed to handle the millions of potential users and should easily cope with these extreme numbers. This seems trivial however existing systems often can't handle these numbers by design. Tarzan requires all users to have knowledge of all other users, which is not practically scalable[13]. TOR only uses a fixed amount of entry nodes and exit nodes[3], which cap the bandwidth enormously when the userbase grows. The systems design should be scalable and performance should not decline with more users.

## 3.3. Attacks

An anonymous Internet should be in some extent resilient to several kinds of attacks. Attacks differ in size, approach, severity and danger. There are passive attacks, where observing is key, and active attacks in which the attacker actively participates. This section covers the main types of attacks.

### Passive attacks

**Timing attack** It should be impossible for a malicious node to gather any information about where information is coming from and where information is going by timing the request and response times. This attack is very hard to conquer and many systems such as OneSwarm have proven to be vulnerable to timing attacks[13] (also see section 2.2).

**Eavesdropping** Malicious nodes should not be able to gather information when eavesdropping the infrastructure. In an eavesdropping attack the adversary will try to follow requests from nodes to exit nodes with statistical analysis of the traffic flow. Data flowing through the circuits should *always* be encrypted and undistinguishable. It should always be plausible that bits flowing from A to B are relay traffic instead of request/response traffic.

## Active attacks

**Masquerading** It should not be possible for an entity to pretend to be another node. Each node should have private information with which he can sign messages in order to ensure that he is the origin of the message. A forged identity can be disastrous for any distributed system.

**Replay attack** In replay attacks, information is stored by an adversary and used later on (replaying the message). This can be done in badly implemented challenge response authentication but also for timing attacks or sybil attacks.

**Sybil attack** Distributed systems that use a reputation system or voting system of some sort are vulnerable to the Sybil Attack. In this attack great amounts of identities are formed and these identities work together. The more identities are collaborating the more influence the whole gains in order of determining the reputation or rating of content and / or users. The Sybil Attack is hard to counter. Making it expensive (in terms of computing power) to enter the distributed system and gain an identity could prevent creating large amounts of identities at once, but this solution is not waterproof. With the budget that organisations such as the NSA have, such attack is easily deployed.

**Eclipse attack** In the Eclipse attack an adversary tries to exclude a node from the network. Often the adversary gives a node false information. When a node requests the state of a system, the malicious node introduces a faulty state which enables him to influence the behavior of the victim node. This can be used to introduce faulty nodes to the victim node, or let the victim node drop connections with 'good' nodes. Once a node is eclipsed, the node is compromised.

## Heartbleed-like attacks

You can build a theoretically unhackable and completely safe algorithm, but as soon as the foundation of the algorithm is hacked, the algorithm itself is useless. HTTPS has been compromised completely due to a bug in the OpenSSL software, see section 1.1. These attacks are difficult to perform and difficult to conquer. Another example of such an attack is cracking a random number generator. The cryptographically secure RSA keys have been compromised by 'listening' to the computer while it generated the keys[25]. In constructing an anonymous Internet it should not be forgotten that if the system relies on something, it should not be assumed that it is unhackable.

## End-device hacking

Whatever protocol is used to display something on the monitor, your privacy is completely gone when somebody sitting next to you can see what you're doing. The same goes if somebody can see what you're doing when he's not even around. End-device hacking can be of great value to an attacker as it is hard to see and nearly impossible to counter. The National Security Agency has successfully implemented

the Cottonmouth series, see figure 3.3. In these Universal Serial Bus (USB) hacks the end-users computers is compromised without him even knowing it.



Figure 3.3: One of the NSA's devices from the ANT Catalogue. Image source: [http://www.spiegel.de/static/happ/netzwelt/2014/na/v1/pub/img/USB/S3223\\_COTTONMOUTH-I.jpg](http://www.spiegel.de/static/happ/netzwelt/2014/na/v1/pub/img/USB/S3223_COTTONMOUTH-I.jpg)

### 3.4. Adversaries

Theoretical attacks on deployed systems only work when the capacity is there to carry out the attack. Attacks often costs a great deal of computational power, money or time. Not everybody has these assets. Adversaries can roughly be divided into three categories; script kiddies, service providers and global adversaries.

#### 'Script kiddie'

Script kiddies is a term sobriquet for single man attackers. These highly vary in intelligence and capability, but can do great harm to systems. A lot of hacks are done by the so called script kiddies. In order to be resilient against script kiddies the distributed network should identify a single entity that is trying to abuse the system. Whenever a single entity tries to forge an identity, to alter messages or block content he should be eliminated from further contact and the network should behave as expected directly afterwards. The problem herein is the immediate recognition of a single malicious entity.

#### Internet service providers

Internet service providers can be summoned to hand information about Internet traffic to law enforcement agencies. Vodafone has recently admitted having wire-taps directly in their infrastructure in some of the countries they are active in<sup>5</sup>.

<sup>5</sup>Guardian news item: [http://www.theguardian.com/business/2014/jun/06/vodafone-reveals-secret-wires-allowing-state-surveillance?CMP=tw\\_t\\_gu](http://www.theguardian.com/business/2014/jun/06/vodafone-reveals-secret-wires-allowing-state-surveillance?CMP=tw_t_gu)

Furthermore Internet service providers be forced to filter data as what happens in Turkey in April 2014 with Twitter<sup>6</sup>. Internet service providers can have control over multiple entities within a distributed network, as multiple users of the network might have the same Internet service providers. These Internet service providers can therefore have more information about data flowing through their infrastructure than a script kiddie. End to end cryptography becomes more and more of a concern when the infrastructure is in control of a 'malicious' party. ISPs are in some way capable of doing collaborative attacks on the system, but will mainly focus on surveillance and content filtering as they can be obliged to do so by law enforcement agencies.

### Global adversary

The most powerful adversaries are the intelligence agencies. The NSA is estimated to have a 10.8 billion dollar budget<sup>7</sup>. Annually, that is. This enables the NSA to deploy the most complicated attacks and develop the most complicated hardware. As the Snowden files indicate it has enabled them to deploy mass surveillance to not only US citizens but citizens worldwide. NSA's SIGAD, also known as PRISM/US-984XN collected data of the biggest Internet websites worldwide, see figure 3.4. Along with PRISM there are multiple other tools developed and used by the NSA, such as XKeyScore. Edward Snowden says its possibilities are endless: *"You could read anyone's email in the world, anybody you've got an email address for. Any website: You can watch traffic to and from it. Any computer that an individual sits at: You can watch it. Any laptop that you're tracking: you can follow it as it moves from place to place throughout the world. It's a one-stop-shop for access to the NSA's information."*<sup>8</sup>. The NSA has access to endless amounts of data by wiretapping underseas Internet cables<sup>9</sup>. Furthermore they have hacked end-devices such as routers and Universal Serial Bus systems (see figure 3.3). It is clear that if an adversary like the NSA wants to exploit or attack a system, they can. It is out of the scope of this thesis to even try to make an anonymous Internet that is 'NSA-proof'.

<sup>6</sup>BBC news item: <http://www.bbc.com/news/world-europe-26677134>



<sup>7</sup>From the "black budget": <http://www.washingtonpost.com/wp-srv/special/national/black-budget/>

<sup>8</sup>From a German interview with Edward Snowden on NDR. Full transcript available here <http://www.commondreams.org/headline/2014/01/27-1>

<sup>9</sup>The Guardian news article: <http://www.theguardian.com/uk/2013/jun/21/gchq-cables-secret-world-communications-nsa>

TOP SECRET//SI//ORCON//NOFORN

Gmail facebook Hotmail Google YAHOO! skype paltalk.com YouTube AOL mail

 (TS//SI//NF) **PRISM Collection Details** 

Current Providers

What Will You Receive in Collection (Surveillance and Stored Comms)?  
It varies by provider. In general:

- Microsoft (Hotmail, etc.)
- Google
- Yahoo!
- Facebook
- PalTalk
- YouTube
- Skype
- AOL
- Apple

- E-mail
- Chat – video, voice
- Videos
- Photos
- Stored data
- VoIP
- File transfers
- Video Conferencing
- Notifications of target activity – logins, etc.
- Online Social Networking details
- **Special Requests**

Complete list and details on PRISM web page:  
Go PRISMFAA

TOP SECRET//SI//ORCON//NOFORN

Figure 3.4: One of the leaked top secret slides of the PRISM presentation, full slides available here [http://www.lemonde.fr/technologies/article/2013/10/21/espionnage-de-la-nsa-tous-les-documents-publies-par-le-monde\\_3499986\\_651865.html](http://www.lemonde.fr/technologies/article/2013/10/21/espionnage-de-la-nsa-tous-les-documents-publies-par-le-monde_3499986_651865.html)



# 4

## Design of p2p onion routing

We utilize the basic idea of TOR, optimized for peer to peer traffic and stripped of all central servers. TOR is for web browsing, the p2p onion router can be used for all kinds of UDP traffic. Circuits are built using the same general method as in TOR. For maximizing throughput AES onion routing is used communicating from originator to exit node. Each participating node in the p2p onion router is a voluntary active relay and possible exit node.

### 4.1. General outline

A node that wants to download a torrent anonymously creates multiple multi-path circuits incrementally in such a way that each node only knows the next hop in the circuit and the previous node in the circuit, but no more. Each node in the circuit proposes a set of next hops, the candidate list. The originator of the circuit decides one. When these circuits are built, data requests are sent through the circuits. The last node in the circuits does the data request and propagates the data back over the circuit.

### Cryptography in the p2p onion router

The p2p onion router uses Elliptic Curve Elgamal for asymmetric encryption[18, 26], Diffie Hellman for session key negotiation[9]. Packet encryption is done with symmetric encryption with 128bit AES . This is implemented in both ECB mode and 'opportunistic cryptography' mode (see session 4.5)[8]. The use of these cryptographic functions is described in 4.3.

### Building circuits

Each process running the p2p onion router protocol has an IP-address, a port and public key (see 4.3). The public key is assumed to be globally accessible through the existing distributed network Dispersy[27]. Every data request used in the p2p protocol is done over several hops instead of directly. These so called *circuits*

are constructed incrementally using `CREATE`, `CREATED`, `EXTEND` and `EXTENDED` packets, just as in TOR. Each circuit is built as follows:

1. The initiator Alice sends a `CREATE` packet to an arbitrary node Bob which she can communicate with. The payload of this `CREATE` packet is the first part of the Diffie Hellman handshake, encrypted with the public key of Bob. This complete packet is also encrypted with the public key of Bob. Section 4.2 contains detailed information about the packets.
2. Upon receiving a `CREATE` packet, Bob calculates the shared Diffie Hellman secret and derives a 'session key' from it. Bob responds with a `CREATED` packet. This packet contains the second part of the Diffie Hellman handshake and a list of nodes which he can communicate with, the candidate list. The list is encrypted with the calculated session key and the entire `CREATED` package is encrypted with the public key of Alice.
3. When Alice cannot decrypt the candidate list using the session key (calculated the same way as Bob calculates it), the circuit is immediately dropped as the identity of Bob can not be guaranteed. When Alice can decrypt the candidate list, Alice and Bob share a secret which they can use for encrypting packets.
4. Alice picks a next hop Charlie from the candidate list and sends an `EXTEND` packet to Bob. This packet contains the identifier of Charlie and the first part of the Diffie Hellman handshake that Alice wants to do with Charlie. The Diffie Hellman handshake part is encrypted with the public key of Charlie and the entire packet is encrypted using AES and the secret that Alice and Bob share.
5. Upon receiving the `EXTEND` packet, Bob sends a `CREATE` packet to Charlie just as Alice sent a `CREATE` packet to Bob, but with the Dhandshake part from Alice.
6. Bob receives a `CREATED` packet from Charlie but can't read the candidate list as it's encrypted with the secret that Alice and Charlie share. It sends an `EXTENDED` packet with the exact same payload of the `CREATED` cell, encrypted with the session key between Alice and Bob.
7. When Alice receives the `EXTENDED` packet she checks if she can decrypt the candidate list. If not the circuit is terminated.
8. When Alice wants to extend the circuit further it wraps the `EXTEND` circuit in layers using AES with as key the session key of the hops, up to the current endpoint of the system, so only that node can read the `EXTEND` packet and other nodes will just forward it as a normal `DATA` packet (see below). The `EXTENDED` packet is only readable by the Alice, so is forwarded by all other nodes.



## Data transmission

Once a circuit is created data can be transmitted over the circuits using `DATA` packets. Each hop knows the direction of the circuit (towards the initiator of the circuit or towards the endpoint of the circuit). During the creation of the circuits each node obtained a session key between itself and the initiator of the circuit, not knowing who the initiator is. When a node receives a `DATA` packet it looks up the direction of the packet. If it's heading towards the endpoint it *decrypts* using AES with the session key that the node has with the circuit. If the packet is heading towards the initiator, the node *encrypts* the data using AES with the session key that the node has with the circuit and forwards the packet. This way only the endpoint of the circuit can read a data request from the originator, which wraps the packet in AES layers using the session keys it has with the nodes in reversed order. Only the originator can read the data that the endpoint sends back through the system by decrypting all layers of AES with the session keys in the order of the hop number of the nodes.

1. Alice wants to request  $x$  over circuit Bob - Charlie - Dave. She wraps the request in AES layers and sends  $\text{END}(\text{END}(\text{ENC}(x)))$  to Bob.
2. Bob peels its AES layer and sends the message  $\text{ENC}(\text{ENC}(x))$  to Charlie.
3. Charlie peels its AES layer and sends the message  $\text{ENC}(x)$  to Dave.
4. Dave requests  $x$  to the server and sends the response encrypted with his AES layer  $\text{ENC}(\text{response})$  to Charlie.
5. Charlie adds his AES layer and sends the message  $\text{ENC}(\text{ENC}(\text{response}))$  to Bob.
6. Bob adds his AES layer and sends the message  $\text{ENC}(\text{ENC}(\text{ENC}(\text{response})))$  to Alice.
7. Alice peels each AES layer and reads the response.

## 4.2. Packet specification

There are seven different kind of packets.

**CREATE** Used for adding single nodes to an existing or new circuit.

**CREATED** Acknowledgement for `CREATE` message.

**EXTEND** Used to tell the last node in the current circuit to add a node.

**EXTENDED** Acknowledgement for `EXTEND` message.

**DATA** Data transfer.

**PING** For keeping circuits alive as UDP is connectionless.

**PONG** Response to `PING` message.

	Description	Size in bytes
Packet Header	Circuit ID	4
Packet Content	Packet type	1
	First part of Diffie Hellman	256

Figure 4.1: The CREATE packet

The `CREATE` packet consist of a packet header (circuit identifier) and packet content. The packet content consist of the packet type and the first part of the Diffie Hellman handshake  $g^x \bmod G$ .

4

	Description	Size in bytes
Packet Header	Circuit ID	4
Packet Content	Packet type	1
	Encrypted second part of Diffie Hellman	256
	Candidate list	variable

Figure 4.2: The CREATED packet

The `CREATED` packet consist of a packet header (circuit identifier) and packet content. The packet content consist of the packet type and the second part of the Diffie Hellman handshake  $g^y \bmod G$  and a candidate list. The candidate list is specified as follows:

Description	Size in bytes	
Number of candidates	1	(per candidate)
Public key	256	

Figure 4.3: The candidate list

The candidate list in a `CREATED` packet is by default encrypted with AES and the session key that the author of the candidate list and the originator of the network share.

	Description	Size in bytes
Packet Header	Circuit ID	4
Packet Content	Packet type	1
	Extend with	8
	Encrypted first part of Diffie Hellman	256

Figure 4.4: The EXTEND packet

The `EXTEND` packet consist of a packet header (circuit identifier) and packet content. The only possible recipient of an `EXTEND` packet is the last node in the circuit. The packet content consist of the packet type, the node which the circuit is to be extended and the first part of the Diffie Hellman handshake  $g^x \bmod G$ . The first part of the Diffie Hellman handshake is encrypted with the public key of the node to be added to the circuit, so only that node can read it.

	Description	Size in bytes
Packet Header	Circuit ID	4
Packet Content	Packet type	1
	Encrypted second part of Diffie Hellman	256
	Candidate list	variable

Figure 4.5: The EXTENDED packet

The `EXTENDED` packet consist of a packet header (circuit identifier) and packet content. The only possible recipient of an `EXTENDED` packet is the owner (initiator) of the circuit. The packet content consist of the packet type, the second part of the Diffie Hellman handshake  $g^y \bmod G$  and a candidate list. The candidate list is encrypted and specified above.

	Description	Size in bytes
Packet Header	Circuit ID	4
Packet Content	Packet type	1
	Address	8
	Data	1024

Figure 4.6: The DATA packet

The `DATA` packet consist of a packet header (circuit identifier) and packet content. The only possible recipient of a `DATA` packet is the owner (initiator) of the

circuit and the last node of the circuit. The packet content consist of the packet type, the address that the exit node should do a request to, and the data. The data field consist of LibTorrent data when a `DATA` packet is heading to the endpoint and the requested data when the packet is heading to the originator of the circuit.

	Description	Size in bytes
Packet Header	Circuit ID	4
Packet Content	Packet type	1

Figure 4.7: The PING packet

## 4

UDP is a connectionless protocol. In order to know which nodes are still active circuits will have to be kept alive. This is done using `PING` messages. At a set interval a `PING` message is sent over the circuit. It is relayed by intermediate nodes and only the last node in the circuit can decrypt the `PING` message. It replies with a `PONG` message to let the originator of the circuit know that the circuit is still alive and well.

	Description	Size in bytes
Packet Header	Circuit ID	4
Packet Content	Packet type	1

Figure 4.8: The PONG packet

### 4.3. Packet encryption

Received packets are always partially encrypted. The packet header is unencrypted and consist only of the circuit identifier. The packet content is always encrypted. The way the recipient of a packet decrypts the packet content depends on what the recipient knows about the circuit on which the packet was received. Public key encryption is done using Elliptic Curve Elgamal encryption. Diffie Hellman data  $g^x \bmod G$  and  $g^y \bmod G$  are calculated randomly per hop per circuit with  $g = 2$  and  $G =$  the 2048 bit prime defined in RFC3526[28]. AES encryption and decryption is done with the first 128 bits of the SHA-1 hash of the shared Diffie Hellman secret.

#### Diagrams

Figure 4.9 shows the packet flow when creating a circuit and transmitting data through the circuit. Details about the packets is described in 4.2. In this diagram,  $\text{Bob}(x)$  means  $x$  encrypted with Bob's public key. `DH-DATA` is the Diffie Hellman data described in 4.2. `ENC(x)` means  $x$  encrypted with AES with the keys described in 4.3. Figure 4.10 shows the transmission of data over a circuit. In this diagram the same syntax is used.

From	To	Packet
Alice	Bob	circ1, Bob(CREATE, Bob(DH-data))
Bob	Alice	circ1, Alice(CREATED, DH-data, ENC({candidates}))
Alice	Bob	circ1, ENC(EXTEND, Charlie, Charlie(DH-data))
Bob	Charlie	circ2, Charlie(CREATE, Charlie(DH-data))
Charlie	Bob	circ2, Bob(CREATED, DH-data, ENC({candidates}))
Bob	Alice	circ1, ENC(EXTENDED, DH-data, ENC({candidates}))
Alice	Bob	circ1, ENC(ENC(EXTEND, Dave, Dave(DH-data)))
Bob	Charlie	circ2, ENC(EXTEND, Dave, Dave(DH-data))
Charlie	Dave	circ3, Dave(CREATE, Dave(DH-data))
Dave	Charlie	circ3, Charlie(CREATED, DH-data, ENC({candidates}))
Charlie	Bob	circ2, ENC(EXTENDED, DH-data, ENC({candidates}))
Bob	Alice	circ1, ENC(ENC(EXTENDED, DH-data, ENC({candidates})))

Figure 4.9: Flow diagram for creating a circuit

From	Packet	To
Alice	Bob	circ1, ENC(ENC(ENC( address, LibTorrent-data)))
Bob	Charlie	circ2, ENC(ENC( address, LibTorrent-data))
Charlie	Dave	circ3, ENC( address, LibTorrent-data)
Dave	address	Normal data request
address	Dave	Normal data response
Dave	Charlie	circ3, ENC( address, DATA )
Charlie	Bob	circ2, ENC(ENC( address, DATA ))
Bob	Alice	circ1, ENC(ENC(ENC( address, DATA )))

Figure 4.10: Flow diagram for transmitting data through a circuit

## 4.4. Block cipher mode of operation

When encrypting data with AES, blocks of 128 bits are encrypted using the session key. Identical parts in the plaintext result in identical parts in the ciphertext. In order to prevent certain structures or repetitions in data to be visible to an eavesdropper encryptions can be made dependant on (all) previous encryptions in the data stream. Figure 4.11<sup>1</sup> gives a clear explanation why electronic codebook mode, in which there is no dependency on the previous data, should be avoided.

There are different modes of encrypting a datastream. Electronic Code Book (ECB) mode is the simplest form in which encrypting a block does not rely on the encryption of any other blocks. The three most common used are:

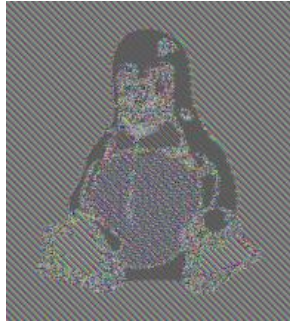
- **Electronic Code Book (ECB)**

There is no relation between the blocks and all blocks can be decrypted with-

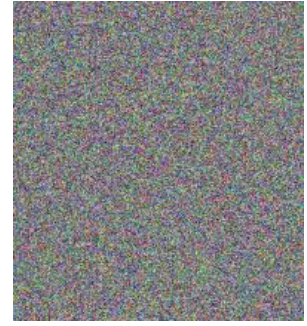
<sup>1</sup>This image is derived from File:Tux.jpg, and therefore requires attribution. All uses are permitted provided that Larry Ewing, the owner of the original image, who requires that you mention him, his email address, lewing@isc.tamu.edu, and The GIMP, according to <http://www.isc.tamu.edu/~lewing/linux/>.



(a) Original image



(b) Encrypted with ECB mode



(c) Encrypted with other mode

Figure 4.11: Comparison of encryption methods for encrypting the Linux Mascot, Tux

4

out any knowledge about order. This should be avoided as visible in figure 4.11.

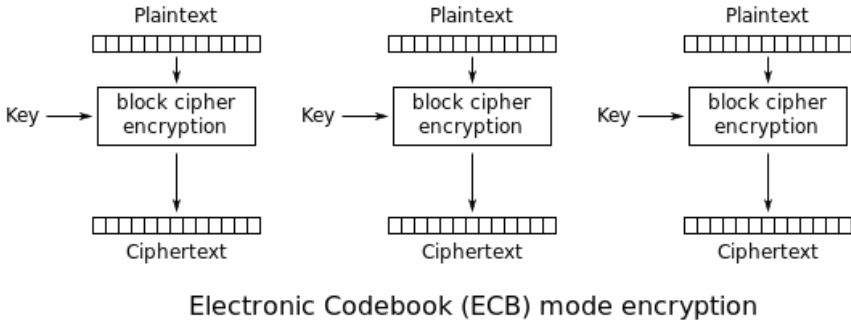


Figure 4.12: ECB encryption

- **Cipher Block Chaining (CBC)**

Block  $x$  that is to be encrypted is XOR-ed with encrypted block  $x - 1$ . Must be encrypted in order but can be decrypted out of order as it can be XOR'ed later on. Relies on an initialization vector which must be the same in encrypting and decrypting. CBC cannot be used with packet loss as  $x - 1$  is not guaranteed to be available when decrypting packet  $x$ .

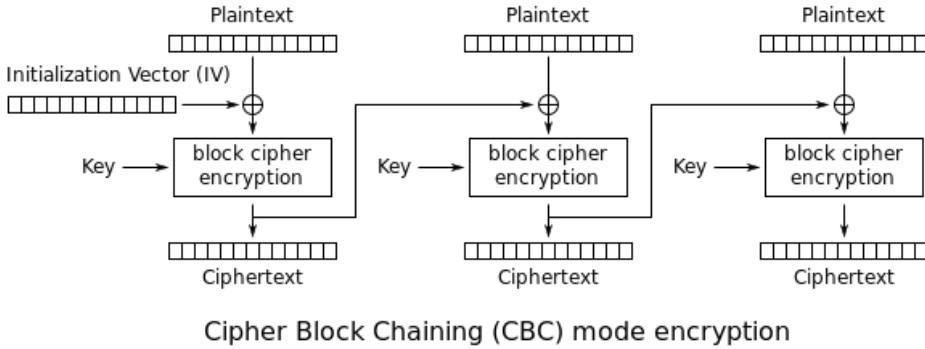


Figure 4.13: CBC encryption

- **Counter (CTR)**

Block  $x$  and  $x + 1$  are encrypted with an extra input; a counter. This counter is usually increased with 1 with each block. Decryption does not rely on previous nodes, but on the counter. Requires either the order to be correct or order number guessing. CTR cannot directly be used with packet loss. It can be used when the counter is guessed. UDP packet size is dynamic, so guessing the counter can be quite cumbersome.

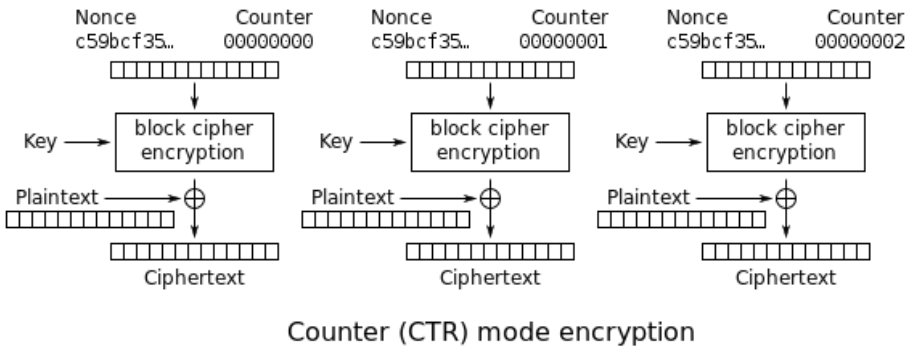


Figure 4.14: CTR encryption

## 4.5. Opportunistic decryption

We use the basic idea of CTR mode and altered it, keeping a counter per UDP packet instead of AES block, in order to allow packet loss and packet reordering. This way we counter pattern leakage with ECB mode but still allow packet loss. The model is called *opportunistic cryptography* or more accurate *opportunistic decryption*. In opportunistic decryption the receiver of a packet *guesses* a key with which

to decrypt the packet.

The underlying technology in p2p onion routing is LibTorrent (see chapter 5). This framework does not ensure packages arriving in order but does handle retransmission of missing UDP packets. These packets can be up to 65.535 bytes and thus consist of multiple AES blocks.

We encrypt these packets independently with CBC. Identical unencrypted packets result in different encrypted packets as we use a counter that increases per packet which defines the key and we use CBC mode within the packets. This way no pattern leakage is present. Figure 4.17 shows the flow diagram of the encryption of a UDP packet. Figure 4.15 shows the simplified encryption of a UDP packet. Figure 4.18 shows the flow diagram of the decryption of a UDP packet. Figure 4.16 shows the simplified decryption of a UDP packet.

In order for this model to work, both parties must agree on the *nonce*, which function as offset and an initialization vector. The nonce will be the first 128 bits of the SHA-1 hash of the shared Diffie Hellman secret. The initialization vector can be any value, 128 bits representing zero suffices. The key used to generate the inner key is the shared Diffie Hellman secret.

## 4

### Sending UDP packet $x$

1. A key for this packet  $k_{udp}$  is generated by encrypting the nonce incremented with the counter  $x$ . It is encrypted using AES 128 bit, with the session key as key.
2. The UDP packet is split in parts of 128 bits.
3. Nonce +  $x$  is appended to the packet.
4. The UDP packet parts and the nonce part are encrypted using the generated key for this packet  $k_{udp}$  and the initialization vector using AES 128 bit in Cipher Block Chaining mode.
5. The encrypted UDP packet parts are merged into one big encrypted packet.
6. The encrypted UDP packet is given to the transport layer.



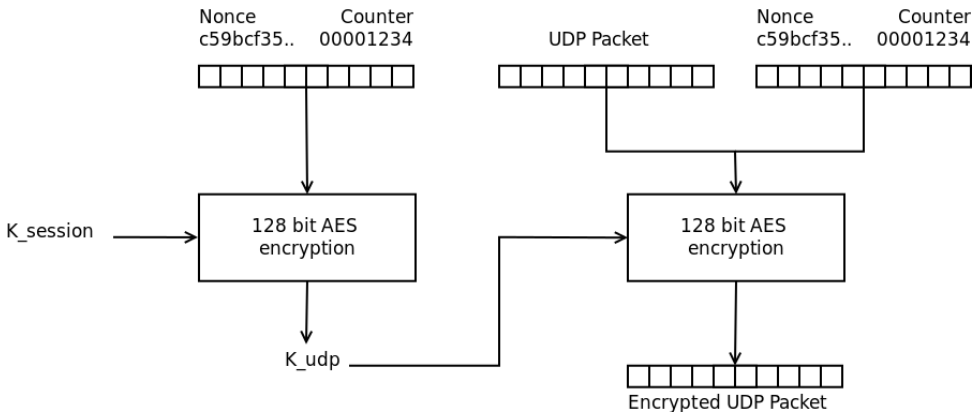


Figure 4.15: Simplified diagram of encrypting the inner onion layer of packet 1234. Notice the Cipher Block Chaining in the second AES block.

### Receiving UDP packet while expecting packet $x$

1. A key for this packet  $k_{\text{udp}}$  is generated by encrypting the nonce incremented with the expected counter  $x$ . It is encrypted using AES 128 bit, with the session key as key.
2. The encrypted UDP packet is split in parts of 128 bits.
3. The encrypted UDP packet parts are decrypted using the generated key for this packet  $k_{\text{udp}}$  and the initialization vector using AES 128 bit in Cipher Block Chaining mode.
4. The unencrypted UDP packet parts are merged into one big unencrypted packet.
5. If the last part of the unencrypted packet does not match the used nonce + the counter  $x$ , there is a retry with  $x+1, x+2 \dots x+9$  or  $x \in \{\text{unreceivedpackets}\}$ .
6. if the unencrypted UDP packet number  $x$  is higher than the previously received packet, missing packets are added to  $\{\text{unreceivedpackets}\}$ .
7. The unencrypted UDP packet is given to LibTorrent.

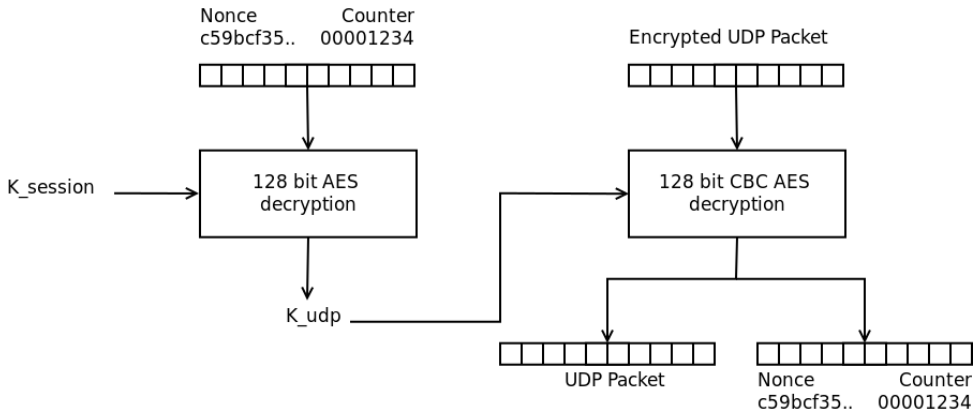


Figure 4.16: Simplified diagram of decrypting the inner onion layer of packet expected to be number 1234. Notice the Cipher Block Chaining in the second AES block.

This model of AES encryption is only necessary in one onion layer, namely the inner one. If the inner onion layer is encrypted in this fashion, the other onion layers can be encrypted with Electronic Code Book mode, as there supportive layer already handles pseudo-randomness. Packet loss will result in a counter mismatch which is followed by counter guessing. The following few packets will be matched against the missing counter as order of messages may vary. If this window becomes too big, the packet is assumed to be missing. These missing packets result in missing parts for LibTorrent, which will redo the request for the part.

In order to check at the receivers side if the counter used to encrypt is as expected, the counter is added to the plain UDP packet before encryption. After decryption with a certain key, the used counter can be checked. If the counter is identical, the packet is decrypted with the right key. If not, the counter must be found.

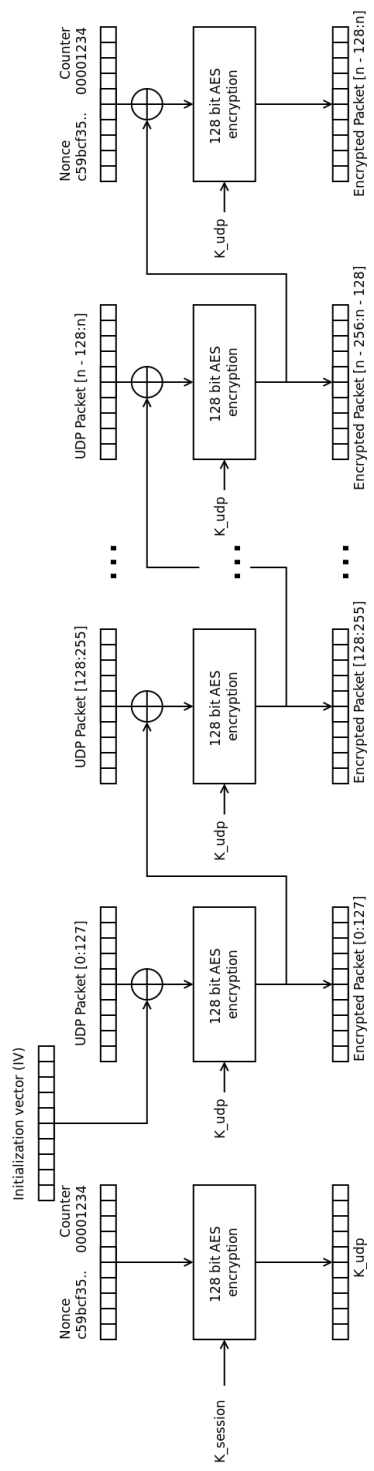


Figure 4.17: Flow chart of the AES encryption per UDP packet. This is the encryption of packet 1234.

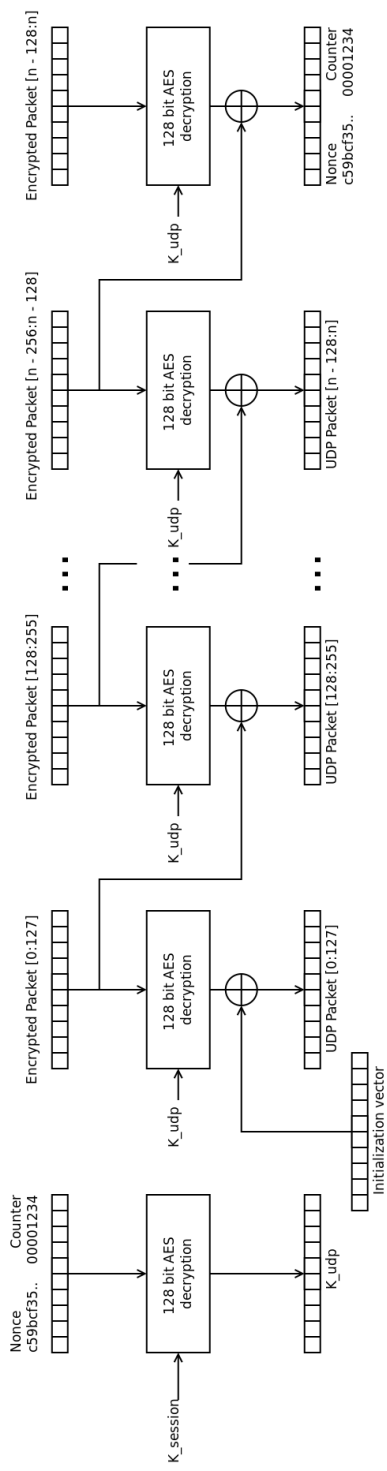


Figure 4.18: Flow chart of the AES decryption per UDP packet. This is the expected to be packet number 1234.



# 5

## Implementation and experiments

The protocol is implemented on the application layer, in Python 2.7. The implementation was a collaborative effort with Chris Tanaskoski[29], with overlapping main responsibilities. Chris Tanaskoski provided a base for circuit creation which is used. Along with the addition of cryptography the underlying base radically changed as well. Development has largely overlapped as our code merged together. It is implemented alongside the peer 2 peer file sharing program Tribler[30], developed at the University of Technology, Delft. An extension to this program is implemented, creating an "Anonymity" community in which users create the circuits and utilize each other. The layered model for the complete implementation is shown in figure 5.1.

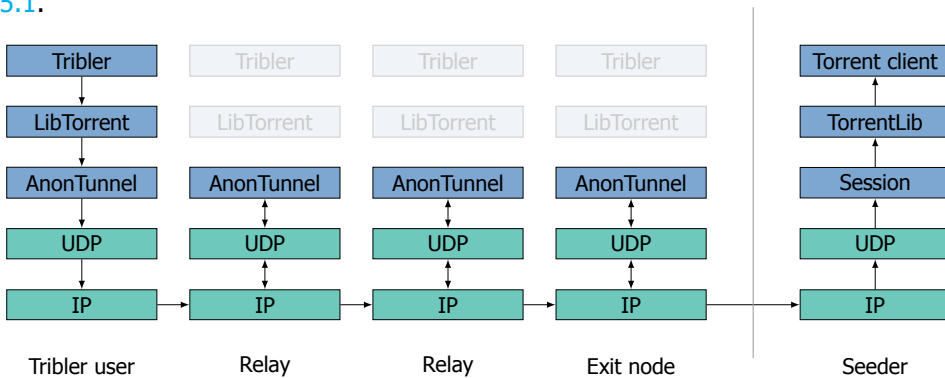


Figure 5.1: The layered model for the p2p onion router as implemented. The arrows show a request over a tunnel of three hops. The machine not in the p2p onion router (the seeder) can have any server such as Socks5, any torrent library such as LibTorrent or LibSwift and any torrent client such as Tribler or  $\mu$ torrent. In the response all arrows will be mirrored.

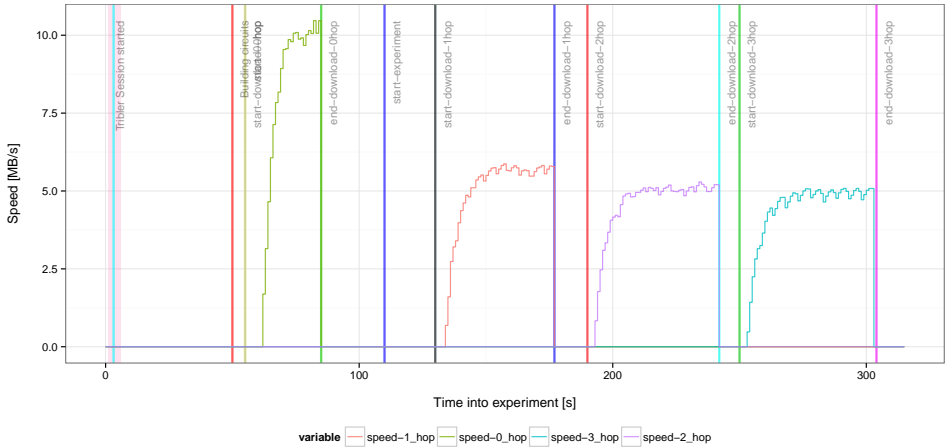


Figure 5.2: The download speed without cryptography without circuits, with 1, 2 and 3 hop circuits deployed on Gumby. Speed decreases with increasing circuit size.

## 5.1. Synthetic experiments

*Gumby* is an experimental framework developed at Delft University of Technology at the department of Parallel and Distributed Systems. The p2p onion router is implemented in Python and tested on Gumby. The Gumby framework can be executed on the Distributed ASCII Supercomputers 4 (DAS4). The p2p onion router is deployed on 74 dual core 2.4GHz 24GB RAM nodes, at the Vrije Universiteit in Amsterdam.<sup>1</sup> On this supercomputer scenarios are deployed in which nodes start building circuits and download a test file. Analysis of outcome of these experiments during the implementation and design of the protocol gave essential information on bottlenecks and design issues. The generated data gives insight on the working of the protocol, the possibilities of the protocol and the downsides of the protocol.

When one hop is added to the circuit, every packet flowing through the circuit has to be encrypted one time extra by the originator and the reply has to be decrypted one time extra. In a CPU limited environment adding hops can cap the performance of the system. It is important to know how much influence the hop size has on the speed of the test download. The following scenario is tested on Gumby.

1. 1000 nodes are started
2. One node creates a one node circuit
3. One node creates a two node circuit
4. One node creates a three node circuit

<sup>1</sup>more info: <http://www.cs.vu.nl/das4/clusters.shtml>

5. A latency test is done
6. 250 MB is downloaded over the one hop circuit
7. 250 MB is downloaded over the two hop circuit
8. 250 MB is downloaded over the three hop circuit

A graph of the established speed of the test download with different circuit lengths is shown in figure 5.3. Dispersy need to be initialized, which is why the circuits are built after 130 seconds. In this experiment there was *no* packet loss. Furthermore, the *ECB* cryptographic module was enabled.

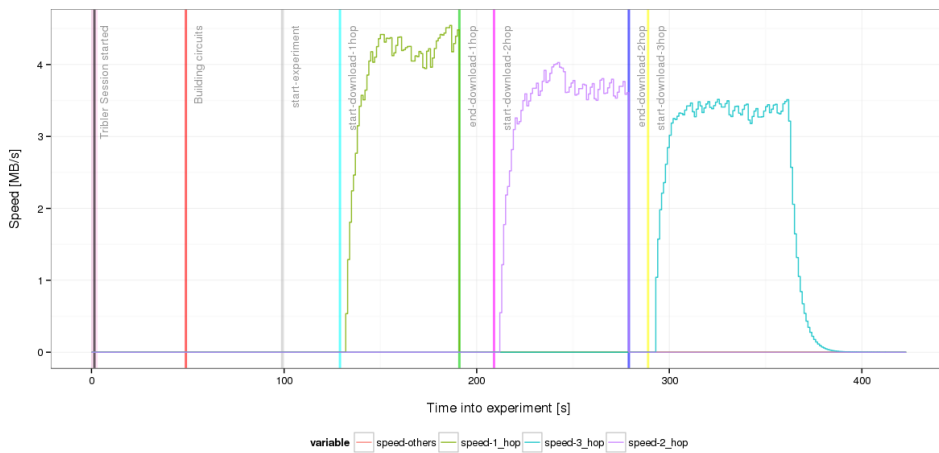


Figure 5.3: The download speed with cryptography with 1, 2 and 3 hop circuits deployed on Gumby. Speed decreases with increasing circuit size.

Figure 5.2 shows the download speeds without cryptography. A download with direct connection to the swarm leads to over 10MB/s. A one hop download without cryptography reaches about 6MB/s, two hops about 5.2MB/s and three hops about 4.8MB/s.

With cryptography enabled, the download with one hop reaches a maximum of about 5MB/s. The two hop download speed decreases to roughly 4.5MB/s and the three hop (anonymous) download speed reaches 4MB/s. This is due to the CPU usage of the downloading node. When the circuit length increases so does the number of cryptographic operations that have to be calculated. Figure 5.4 shows that with the one hop download the CPU of the downloading node is running at full capacity. There is no more room for extra cryptography, so a performance drop is inevitable when the circuit length increases.

Added cryptography leads to a decrease in download speed of about 20%. This is due to the extra load on the CPU when cryptography is enabled. Instead of

simply interpreting or forwarding the data, the data has to be encrypted and decrypted. This drop in performance still leads to download speeds usable for HD video streaming.

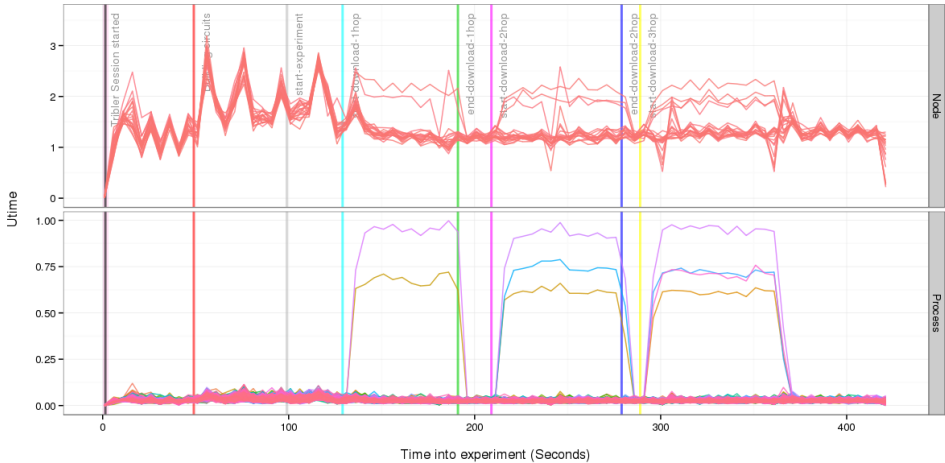


Figure 5.4: The CPU usage of the download deployed on Gumbo. Lower graph clearly shows full CPU usage on the node downloading the test file.

The round trip time of a packet is shown in figure 5.5. This latency test is conducted by sending 100 packets over each circuit and measuring the difference in time between the request and the response. The mean round trip time for a one hop circuit is 1.88ms. For a two hop circuit this is 2.38ms and for a three hop circuit 3.25ms. Some packets take exceptionally longer where most packets arrive before the average time. The 50th percentile is 0.95ms with a one hop circuit, 1.53 with a two hop circuit and 2.08ms with a three hop circuit. With a longer circuit the latency is less predictable as the slope of the graph gets flatter with increasing circuit length.

### Packet loss in onion routing

Optimally packets that are somehow lost are retransmitted by the sender when no acknowledgement is sent. Packets can get lost by faulty network architecture, congestion on the network, bugs in network drivers or routing failures. Normally there is no packet loss, or packet loss is handled by the network layer. In stream encryption this however forms a problem as stated in section 4.5. When using onion routing, the packet loss gets worse.

Consider a packet loss of 0.1%. In the following examples we consider a uniform distribution of lost packets. This should not be any problem when A and B are communicating; only one out of a thousand packets are dropped. Consider the circuit A-B-C-D-Endpoint. Every data packet now has to be sent from A to B, from B to C, from C to D, from D to the Endpoint and all the way back. With a packet loss of 0.1% this aggregates to  $1 - (0.999)^8 = 0.8\%$  packet loss. With a packet loss of



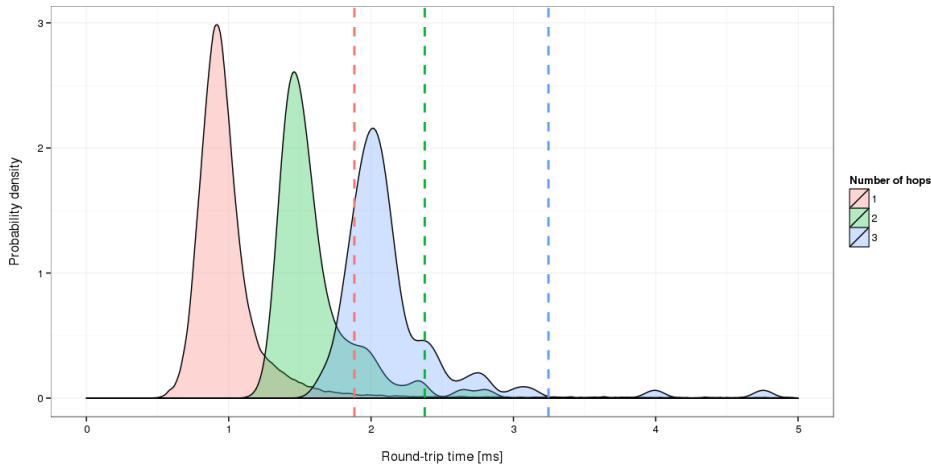


Figure 5.5: Latency test with three circuit lengths.

1%, which is high enough for most UDP applications to fail[31]<sup>2</sup>, this aggregates to  $1 - (0.990)^8 = 7.7\%$  packet loss.

With building circuits this is disastrous as no retransmission is implemented. Consider building a three hop circuit. Without retransmission, a circuit is built using three `CREATE` messages, three consecutive `CREATED` messages, two `EXTEND` messages, two `EXTENDED` messages and two `DATA` forwards. With twelve packets traveling the network, packet loss becomes a serious problem. When there is 0.1% change of packet loss, this aggregates to  $1 - (0.999)^{12} = 1.1\%$  packet loss. This means that 1.1% of the circuit creations fail. Consider a 1% packet loss when creating a circuit. This aggregates to  $1 - (0.990)^{12} = 11.4\%$  packet loss. This means that 11.4% of the circuit creations fail.

Furthermore, when circuits are created, they will have to be kept alive. Because UDP is connectionless, the circuits will have to be pinged overtime when they are idle. A `PING` message travels from the originator to the exit node and a `PONG` message travels back. This means that a one hop circuit ping with 0.1% packet loss has a chance of  $1 - (0.999)^2 = 0.2\%$  chance of failing. With a three hop circuit and a 1% packet loss this increases to  $1 - (0.99)^6 = 5.85\%$ . In order to allow a certain amount of packet loss the ping timeout for circuits is chosen to be greater than three times the ping interval in order to allow two consecutive ping attempts to fail, resulting in  $1 - 0.0585^3$  100% ping success on a 1% packet loss in a three hop circuit. Figure 5.11 shows the aggregated packet loss in creating circuits, pinging circuits and doing data request over those circuits. It is clearly visible that increasing circuit length enormously increases the packet loss as messages will have to a longer path and have a greater chance of getting dropped somewhere along the circuit.

<sup>2</sup>Informal but informative article on packet loss and the effects of packet loss in <http://www.pacificwireless.com.au/packet-loss-and-latency.html>

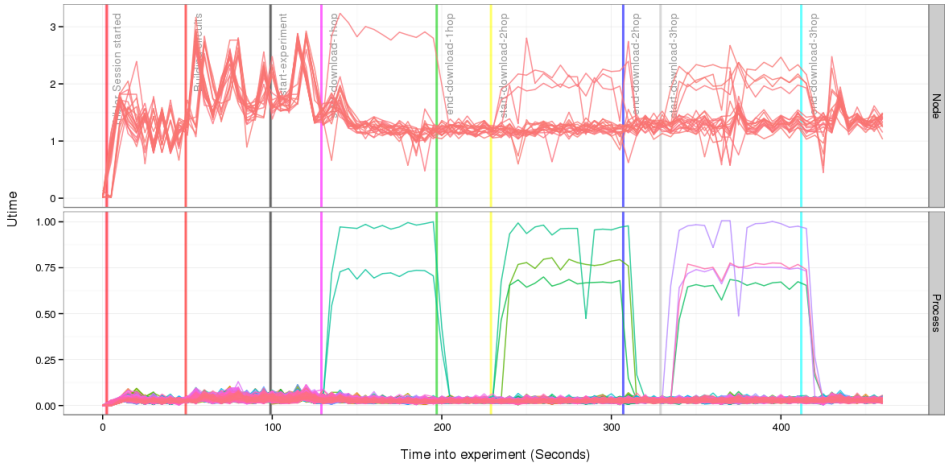


Figure 5.6: CPU usage in opportunistic crypto

5

### Different encryption techniques

The default AES encryption technique (ECB) as described in 4.3 makes packet loss possible as decrypting packet  $x$  does not rely on  $x - 1$ . In opportunistic decryption as described in 4.5, decrypting packet  $x$  does rely on packet  $x - 1$  in the form of a counter. This model is tested by changing the packet loss in different scenarios. This forces counter mismatches in which the nodes have to re-decrypt with a guessed counter. Lost packets will have to be re-requested by LibTorrent. Figure 5.7 shows the download speeds with one hop, two hops and three hops with *no* packet loss, with the opportunistic cryptography module enabled. Figure 5.8 shows the download speeds with one hop, two hops and three hops, with 0.1% packet loss using opportunistic cryptography. Figure 5.9 shows the download speed with one hop, two hops and three hops with 1% packet loss with opportunistic cryptography. Figure 5.10 shows the download speed with one hop, two hops and three hops with 5% packet loss and opportunistic cryptography.

Without packet loss and opportunistic cryptography enabled the speed reaches 4MB/s with a one hop circuit, about 3.5MB/s with a two hop circuit and around 3MB/s with a three hop circuit. Due to key encryption per packet in opportunistic cryptography extra cryptographic computations have to be made, figure 5.6 clearly shows that the CPU is saturated. For one hop the CPU of both the originator and the exit node are saturated. During the two and three hops download the CPU of the relays are also heavily loaded. Figure 5.8 shows that with 0.1% packet loss the one hop download reaches about 3.5MB/s, which is slightly lower than without packet loss. The speed decreases to slightly more than 3MB/s with a two hop circuit and slightly under 3MB/s with a three hop circuit. This bandwidth is still high enough to stream HD videos. With a 1% packet loss the speed dramatically decreases due to the aggregated packet loss of 3.94% with one hop, 5.85% with two hops and 7.73% with three hops. Figure 5.9 shows that the speed with one hop still peaks

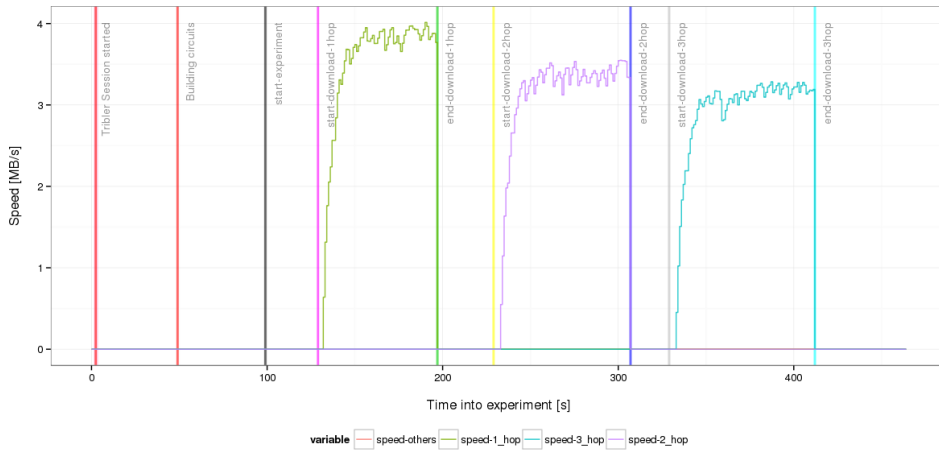


Figure 5.7: Opportunistic cryptography performance with 0% packet loss

at around 3MB/s, but with two hops the download gets only 2MB/s. The three hop circuit peaks at around 1.3MB/s, but fails to get a stable download. It is clear from figure 5.10 that with 5% packet loss onion routing with opportunistic cryptography will fail. The one hop circuit still reaches 1MB/s as peak, but the speed of the two hop download is unstable and peaks at around 150KB/s. The random uniform packet loss is implemented in the endpoint of Tribler, which does not include packet loss from the seeder towards the exit node of the circuits. This exponential decrease of speed with an increasing packet loss is due to LibTorrent. The library notices the packet loss and decreases the speed as a solution in order to decrease the packet loss. Figure 5.12 shows the download speed without circuits but with the same packet loss as aggregated in onion routing. Test downloads are started with 0.8% packet loss, 7.73% packet loss and 33.66% packet loss. This corresponds to the aggregated packet loss of a three hop circuit download with 0.1%, 1% and 5% respectively. The obtained decrease in speed matches the decrease in speeds in figures 5.8, 5.9 and 5.10. There is no cryptography involved so with 0.1% and 7.7% packet loss the speed is higher than in figure 5.8 and 5.9. With 33.7% packet loss the download is completely broken which corresponds with 5% packet loss with three hops in figure 5.10.

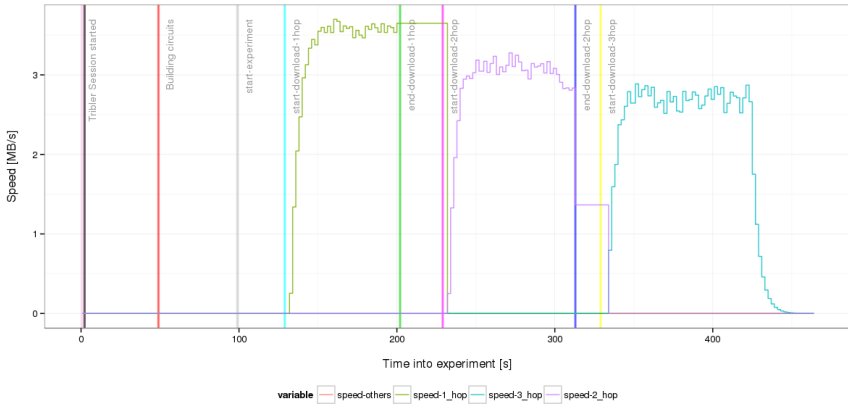


Figure 5.8: Opportunistic cryptography performance with 0.1% packet loss

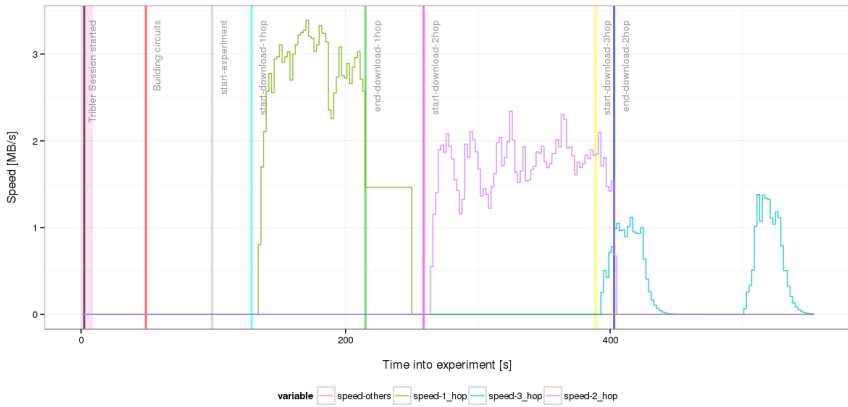


Figure 5.9: Opportunistic cryptography performance with 1% packet loss

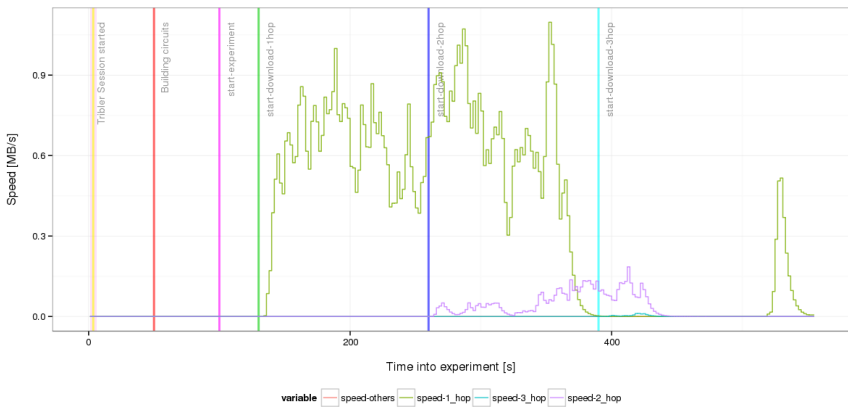


Figure 5.10: Opportunistic cryptography performance with 5% packet loss

Packet loss	0.10	0.50	1.00	2.00	5.00
Creating a 1 hop circuit	0.20	1.00	2.00	3.96	9.75
Creating a 2 hop circuit	0.60	2.96	5.85	11.42	26.49
Creating a 3 hop circuit	1.12	5.84	11.36	21.53	45.96
Pinging a 1 hop circuit	0.20	1.00	2.00	3.96	9.75
Pinging a 2 hop circuit	0.40	1.99	3.94	7.76	18.55
Pinging a 2 hop circuit	0.60	2.96	5.85	11.42	26.49
Data request without hops	0.20	1.00	2.00	3.96	9.75
Data request over 1 hop	0.40	1.99	3.94	7.76	18.55
Data request over 2 hop	0.60	2.96	5.85	11.42	26.49
Data request over 3 hop	0.80	3.93	7.73	14.92	33.66

Figure 5.11: Aggregated packet loss in onion routing (in percentage)

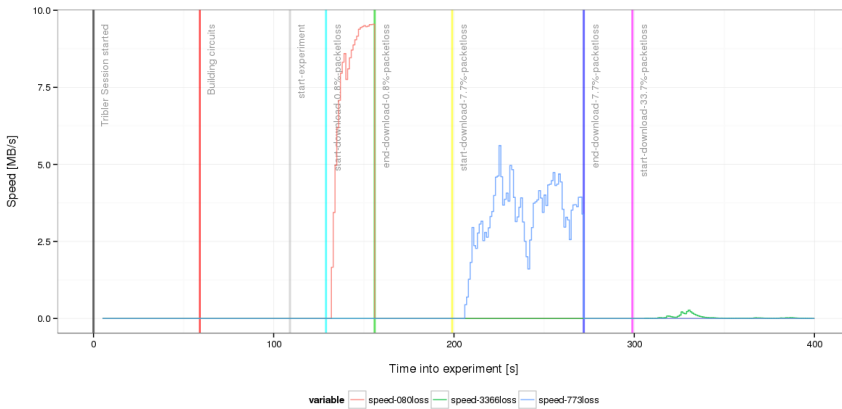


Figure 5.12: Aggregated packet loss tested directly against LibTorrent

## 5.2. Profiling

The protocol is cryptographically intensive and the code must be optimized in order to minimize overhead. This is done by profiling the Python code that is executed. The cryptographic calculations are done using OpenSSL (with the M2Crypto<sup>3</sup> python library) which is written in C++. Profiling led to enormous performance increase with several fixes and it identified the bottlenecks of the system.

Profiling not only showed the performance bottlenecks in number of times methods were executed, but also shows the total time spent within functions. One example of a fix that resulted from profiling is the replacement of the built-in python `pow` method with the superior `gmpy` package `pow` function. Although on some machines the built-in `pow` function of python sufficed for the calculations, on some machines circuits couldn't be created.

Duration	Entry	Average	Count
161.44	Callback_one_task	0.06	7045
11.91	AllChannelConversion._decode_message	0.02	5877
8.58	ElGamalCrypto.is_valid_signature	0.00	4833
2.39	Dispersy_update	0.08	1417
2.23	DispersyDatabase.execute	0.01	22211
1.67	ChannelConversion._decode_message	0.03	318
1.54	DispersyDatabase.commit	0.04	36
0.83	Dispersy_store	0.01	947
0.63	ElGamalCrypto.key_from_public_bin	0.00	2638
0.61	SearchConversion._decode_message	0.02	238
0.51	AllChannelCommunity.dispersy_claim_s...	0.17	3
0.50	ElGamalCrypto.key_from_public_pem	0.00	2638
0.43	ElGamalCrypto.create_signature	0.00	430
0.43	ElGamalCrypto.key_to_bin	0.00	2660

Figure 5.13: Tribler's built-in profiler. Only methods with `@attach_runtime_statistics(u"{0.__class__.__name__}.{function_name}")` are indexed in the profiler

## 5.3. Tribler integration

The p2p onion router is integrated in the p2p file sharing software Tribler. Tribler is developed at the University of Technology in Delft and has evolved through the years from an exploratory project to a smart decentralized peer-to-peer file sharing system. Tribler is chosen as test platform because of the ideology of the Tribler development team and the great collaboration with the team members of Tribler during the development of the protocol.

### 11kloc pull request with 702 commits

Our TOR like enhancement of Tribler is merged with the Tribler software in one GitHub pull request, which consisted of over 11kloc changes in over 700 commits.

<sup>3</sup>More info: <https://pypi.python.org/pypi/M2Crypto>

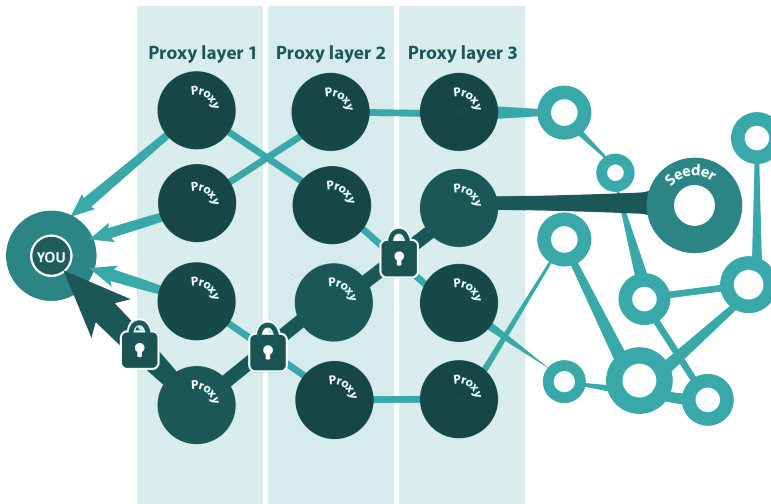


Figure 5.14: Anonymous downloading in Tribler. Image by Myra Vreede

The addition of an anonymity community to Tribler was developed during an intensive code refactoring in the core. This led to great discrepancy between the Anonymity community and Tribler. The software merge was initiated in April 2014 and completed in May 2014.

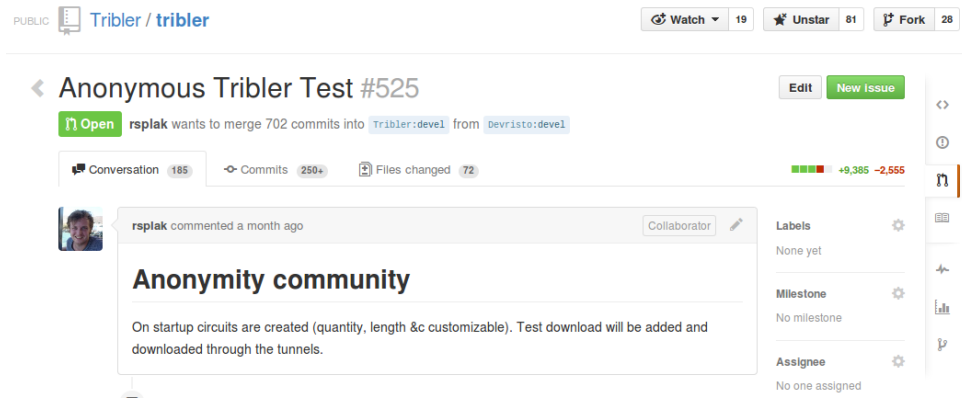


Figure 5.16: The pull request

### 50 MB test

When users launch Tribler four circuits are automatically created using the other 'candidates' in the *ProxyCommunity swarm* in Dispersy. These circuits are idle (and pinged) until the test download starts after five minutes. This is done in order to handle the vast amount of data that would overload the circuits when everybody would start downloading on startup. This way, the first five minutes are purely

based on seeding the test downloads with circuits that are built by other users. After five minutes a 50 MB test download is automatically started over the circuits. There are four static seeders which are queried over the circuits. The 50 MB test download is hosted at four locations within the power of the Tribler software development team to guarantee a high throughput upload. Statistics about the speed of the download are gathered in order to gather global real life statistics of anonymous downloading using the circuits. Users automatically participate in this test download and usage statistics are completely anonymized. Figure 5.14 shows a schematic overview of a test download. Figure 5.15 shows a screenshot of the anonymity panel in a running Tribler instance whilst downloading the test file. When the download is finished users are notified with a pop-up shown in figure 5.17. A screenshot of the way an anonymous downloader is connected to the seeders is shown in figure 5.18.

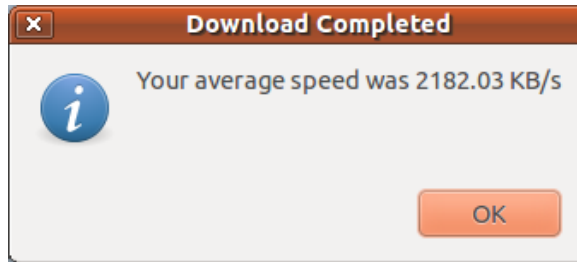


Figure 5.17: When the test download finishes users are informed about the speed of the download.



The screenshot displays the Tribler web interface. At the top, a status bar shows system information: 'm 28% c 51% v 77% b 100% t 13:27:44 d 18/05'. Below this is a search bar and navigation icons. The main content area is divided into several sections:

- Home:** A list of categories with counts: Results (28), Channels (1), Downloads (0), Anonymity (highlighted), and Videoplayer (0).
- Network Graph:** A central visualization of the Tribler network, showing nodes (circles) and connections (lines). A central node is highlighted in red, and a tooltip indicates 'NI = 25'.
- Circuits Table:** A table listing active circuits with their IDs, online status, hops, and bytes up/down.
 

Circuit	Online	Hops	Bytes up	Bytes do
179909	READY	3/3	50.47 KB	2.55 MB
116263	READY	3/3	79.79 KB	4.14 MB
170321	READY	3/3	111.67...	5.69 MB
224923	READY	3/3	97.30 KB	5.12 MB
148851	READY	3/3	0.00 KB	0.00 KB
79449	READY	3/3	0.00 KB	0.00 KB
156373	READY	3/3	0.00 KB	0.00 KB
100657	READY	3/3	0.00 KB	0.00 KB
- Log:** A scrollable log showing the creation and extension of circuits, such as '13:26:42: Extended circuit 170321' and '13:26:52: Created circuit 224923'.

At the bottom right, a 'Family filter' is checked, and system resources are shown: '490 KB/s' and '8.0GB'.

Figure 5.15: When Tribler starts, circuits are automatically created

5

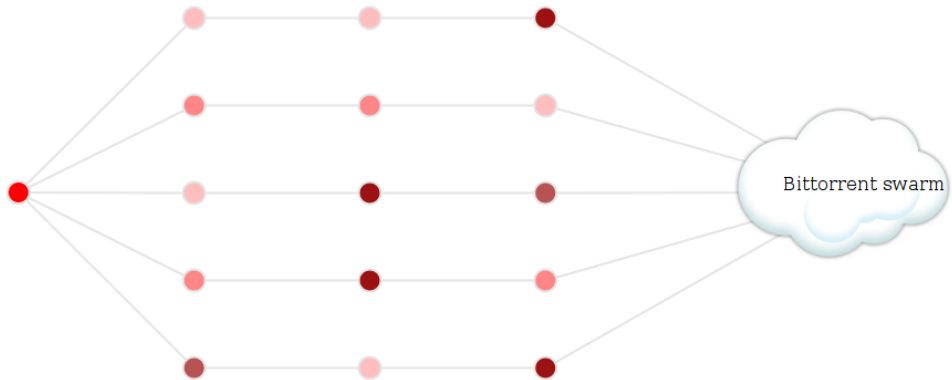


Figure 5.18: A screenshot of the anonymity section in Tribler. Instead of a direct connection from downloader (dot on the left) to the swarm (cloud), the circuits are used to connect to seeders.

# 6

## Future work

The current implementation of the p2p onion router provides anonymity when downloading content. There are still a lot of things to do. It is currently only tested on Linux and Microsoft Windows. As of June 2014 proxies can be deployed on Android devices and anonymous downloading is enabled on Android devices. Full `.torrent` should be implemented after the real world test is deployed. Anonymous video streaming should be implemented and anonymous uploading should be enabled. In the current implementation, downloading is anonymous but seeding is still old-fashioned. This clearly keeps non acceptable content out of the system, as uploaders of illegal content would easily be traceable. For now this is good, but future versions of the software should be able to offer anonymous seeding. This should be thoroughly thought through and is out of the scope of this thesis. There are some easy and well known design principles for anonymous content sharing using onion routing, but there are also some shortcomings of current implementation and system environment that need to be overcome.

### 6.1. Anonymous seeding

The approach TOR uses makes use of *introduction points* and *rendezvous points*. Somebody that wants to offer content creates circuits in the same way as circuits are 'normally' created. The endpoints of these circuits will function as introduction points, to which a user can do a request. The locations of these introduction points are uploaded to the distributed hash table which contains the hidden services, in the form of XYZ.onion where XYZ is a 16 character string generated from the services public key<sup>1</sup>.

A client connects to the introduction point and gives it an address to which the hidden service can connect. This process is done in several steps:

1. The hidden service creates circuits to introduction points and makes them available in the distributed hash table (DHT).

<sup>1</sup>More on TOR hidden services: <https://tor.eff.org/docs/hidden-services.html.en>

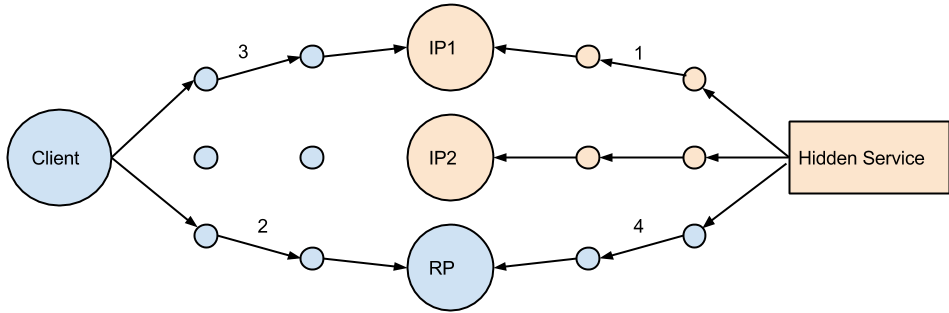


Figure 6.1: The working of hidden services

2. The client creates a circuit to a random rendezvous point.
3. The client connects to the introduction point with information about the rendezvous point, the introduction point forwards it to the hidden service.
4. The hidden service creates a circuit to the rendezvous point, which sends a message to the client that the connection is successful.
5. The client and the hidden service can communicate through the rendezvous point.

## 6

As for the future, anonymous seeding has to be designed and implemented for p2p onion router. Research on anonymous seeding has recently started and is planned to be launched in the end of 2014. A file that is anonymously uploaded *and* anonymously downloaded will have to travel two circuits in order to be untraceable. This will increase the total circuit length and thus the total encryption and decryption CPU cycles for both the uploader and the downloader. This will dramatically decrease the throughput as the speed is CPU limited as shown in section 5. Figure 6.2 shows a schematic overview of anonymous uploading in future Tribler.

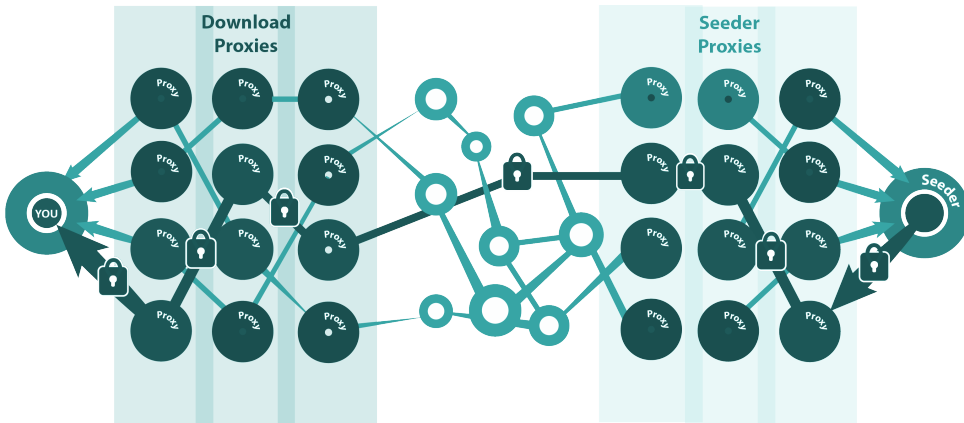


Figure 6.2: Anonymous uploading in Tribler. (Image by Myra Vreede)

## 6.2. Resilience to attacks

In the current implementation the circuits are created incrementally, in which the last hop in the circuit returns a candidate list. The originator of the circuit picks one and extends the circuit. A trivial weakness of the system is the circuit selection. Any malicious node can return a set of malicious nodes and the circuit is compromised. With a fixed circuit length this reveals the identity of the originator. With a variable circuit length, timing attacks can still reveal the originator of the circuit.

In the near future, *Bartercast* will be deployed as a semantic overlay which keeps track of download / upload rate and stability. The original *Bartercast* system only relied on download upload ratio[32] and it is currently being enriched. *Bartercast* will function as a reputation system on which hop selection is based. The originator of the circuit can ask *Bartercast* if any of the candidate nodes are to be trusted and he can base its decision on the reputation of the nodes.

In the current implementation circuit identifiers are plaintext and can easily be linked to each other. With as few traffic as there currently is, it is very easy for a global observer to follow traffic and determine downloader and exit node. With these two linked, an eavesdropper knows exactly what the downloader is doing. In order to counter this form of traffic analysis cover traffic will have to be implemented, which unfortunately will pressure the network load. An approach to counter traffic analysis based on circuit identifier *DTLS* could be introduced as underlying protocol for UDP, which fully encrypts UDP traffic.

Before the software is distributed as an anonymous Internet, the resilience to attacks will have to be investigated. The different kinds of attacks described in 3.3 should be performed and risks and damage should be measured. Only with insight of these attacks the system can be enhanced to counter these attacks.

### 6.3. Circumventing infrastructure

One of the goals of anonymous Internet is to be able to bypass filtering and blockage of traffic by governmental forces. One of the major advantages of the governmental forces is that they have full control of the infrastructure. Therefore the software should be able to function fully on mobile devices. An internet of interconnected devices could be used, or devices connected to infrastructures out of reach for the governmental forces. This could be Telecom carriers, satellites, other ISPs or a combination of providers.

#### Carrier Grade NAT

Telecom providers often use a so called *carrier grade NAT* or *large-scale NAT*, which makes it possible for multiple devices to share the same address space. This introduces the same problem as described in section 3.2, it cannot be certain that if  $A$  and  $B$  can both communicate with  $C$ , they can communicate with each other. When a large portion of the community is behind such a carrier grade NAT, the candidates that a node can find will be fewer than with other NAT's, as routers used by carriers are often very hard to puncture[24]. Figure 6.3 shows a Carrier NAT.

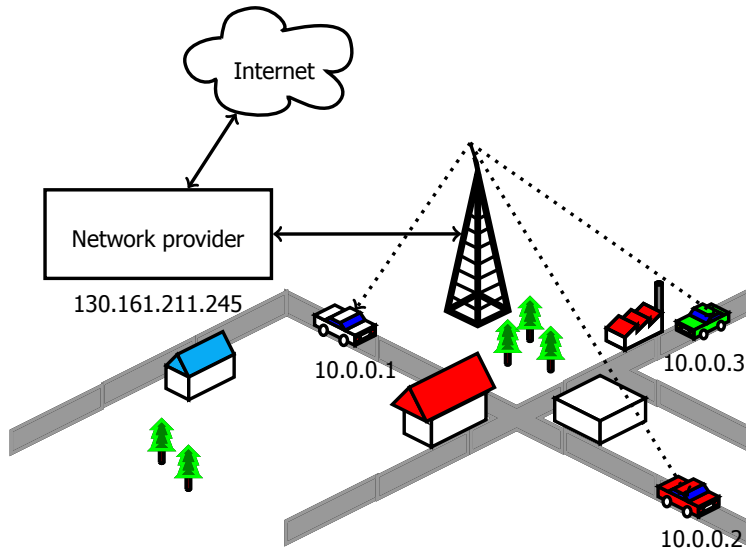


Figure 6.3: Carrier NAT. Users of one provider share the same external IP.

#### Rewarding PORT forwarding

A clear problem that arises with the anonymous seeding model described in 6.1 is that the introduction point and the rendezvous point must be reachable the first hops of both the content requesters circuit as the content provider. The fact that the content provider can build a circuit with an introduction point as end node does not imply that the introduction point is connectible. The same goes for the rendezvous

point picked by the content requester.

In ideal situations the introduction point and the rendezvous point would be fully connectible. So either there shouldn't be any NAT at all or the ports should be opened and forwarded. Port forwarding can't be expected of the average user, so an award system has to be implemented that rewards port forwarding. People that open their ports and are thus suitable to use as an introduction point and rendezvous point could be given some advantages such as the right to cause a higher network load than other users. This however is out of the scope of this thesis.





# 7

## Conclusion

In order to circumvent governmental mass surveillance on the Internet many protocols have been introduced such as TOR, Tarzan and OneSwarm. Only TOR has partially succeeded and has gathered a userbase of over one million daily users. TOR offers anonymity, but speeds peak at only 200KB/s. This is far from enough in the current multimedial era. There is no anonymous Internet which satisfies user expectations such as high-bandwidth and auto-configuration.

This thesis provides a proof-of-concept of an anonymous file sharing network which tries to balance anonymity and user friendliness. The *p2p onion router* is introduced, which takes advantage of both onion routing and peer to peer traffic in order to provide high bandwidth anonymous file sharing. A new cryptographic approach based on CTR mode is introduced. This *opportunistic cryptography* ensures that patterns are not leaked, but does not require packet ordering and the certainty that packets arrive. This makes it suitable for UDP traffic whilst still being resilient to eavesdroppers and traffic analysts.

Synthetic experiments showed that speeds up to 4MB/s can be reached with opportunistic encryption over a three hop circuit. This is fast enough for even HD video streaming. The network still provides up to 3MB/s with 0.1% packet loss. The p2p onion router is fully integrated in the open source file sharing system Tribler. An anonymous download test is implemented in which Tribler users automatically download a 50MB test file over circuits created with other users. Every user functions both as relay and as exit node for other users, which makes the p2p onion router both fast and highly scalable.

The introduced system is however in an early stage in which circuit selection is a weak spot. Before deployment, research must focus on resilience against traffic analysis and other types of attacks. Furthermore it is impossible to anonymously upload files in the p2p onion router. Public opinion upon release of the Tribler version with the anonymous test download will indicate whether being an exit node discourages people from using the p2p onion router, although there is plausible deniability that the content is someone else's.



# Glossary

**Adversary** In the context of distributed networks and cryptography an adversary is an entity that is trying to break the network. The adversary tries to break the cryptographic system, the trust model or influences other entities in the network. More on adversaries in section 3.4.

**Asymmetric Encryption** Mathematical transformation of a value to another value using a key. Decryption is done with another key. Asymmetric encryption is mostly based on the infeasibility of deriving shared prime factors.

**Attacker** See Adversary.

**Bartercast** Reputation system created by the Delft University of Technology. More on Bartercast at <http://tribler.org/BarterCast>.

**BitTorrent** Protocol for Peer to Peer file sharing. Files are downloaded through multiple peers. Peers are rewarded with the tit-for-tat mechanism. More on BitTorrent on <http://bittorrent.org>.

**BitTorrent Swarm** Swarm of nodes that are sharing the same torrent file.

**Certificate authority** Entity which issues SSL certificates. Every web browser trusts a set of *root* certificate authorities to be genuine.

**Client-side** The 'side' of the communication where the user is. In browsing The Internet it's the side of the person who is surfing the web.

**Community Swarm** Swarm of nodes that are in the same community.

**Cottonmouth** Device created by the NSA which hacks a target computer by their Universal Serial Bus. More information in the ANT catalogue: <https://www.eff.org/document/20131230-appelbaum-nsa-ant-catalog>.

**Diffie Hellman** Algorithm to derive shared secrets. Two entities generate a random number and use it as input in a one way function. The output is send to each other. Together with their privately generated random number a shared secret can be derived which is impossible to find by eavesdroppers.

**Dispersy Community** Group of nodes in Dispersy that are interconnected due to the common community identifier.

**Edward Snowden** Former employee of the National Security Agency of the United States of America. Disclosed thousands of classified documents of the USA.

**ElGamal** Asymmetric encryption form based on Diffie Hellman developed by Taher Elgamal. Invented in 1985 but widely used since 2004. Based on the principle that it is infeasible to generate a discrete logarithm of a random elliptic curve element to a known base point.

**Elliptic Curve Cryptography** Form of asymmetric cryptography that is based on the mathematical property of elliptic curves and finite fields.

**End-to-end cryptography** Cryptographic system for which two statements hold when A and B are communicating. B is the only one that can decrypt a message that is sent to him. B can tell with absolute certainty that A is the sender of the message.

**Flash proxy** Attempt to block censorship on the Internet. See section 2.4.

**Glimmerglass** Governmental contractor of the United States of America that produce devices which can tap underwater cables independent of the data rate, protocol and format. More information: <http://www.theatlantic.com/international/archive/2013/07/the-creepy-long-standing-practice-of-undersea-cable-tapping/277855/>.

**Gumby** Synthetic experiment framework developed at the Delft University of Technology. Available open source at <http://github.com/tribler/gumby>.

**Internet Engineering Task Force** Group of volunteers that try to standardize algorithms and methods with the mission to create a safer and better internet. More information available at <http://ietf.org>.

**Leecher** Entity that is downloading (parts of) a file in BitTorrent.

**LibTorrent** C++ implementation of the BitTorrent protocol with full support for TCP, UDP and  $\mu$ TP. Available for download at <http://libtorrent.org>.

**M2Crypto** Python extension which implements OpenSSL cryptographic functionalities. Available for download at <https://pypi.python.org/pypi/M2Crypto>.

**MAINWAY** Program of the NSA which was used to collect records of over 1.9 trillion telephone calls. More information: [http://www.democracynow.org/2006/5/12/three\\_major\\_telecom\\_companies\\_help\\_us](http://www.democracynow.org/2006/5/12/three_major_telecom_companies_help_us).

**Network Address Translation** NAT is used as a firewall and as a way of letting multiple entities make use of the same IP address. The address information in IP packets is interpreted by the NAT and transferred to another IP address.

**OneSwarm** Attempt to anonymize the Internet. See section 2.2.

**Onion Routing** Wrapping a message in multiple 'layers' using asymmetric encryption. When a message is forwarded, a layer is peeled by the private key or added by the private key.

**OpenSSL** Open source implementation of a lot of cryptographic standards including RSA, AES, ElGamal, SHA-1 and many more. More information and download at <http://openssl.org>.

**Perfect Forward Secrecy** Term used in cryptography that ensures that once a private key has been compromised, previous communication cannot be decrypted.

**Plausible deniability** In anonymous Internet context plausible deniability is the ability to deny that data that came by your computer is yours. This is possible for for example TOR exit nodes.

**PRISM** Program of the NSA that focusses on electronic mass surveillance of major Internet companies such as Google.

**Seeder** Entity that is uploading (parts of) a file in BitTorrent.

**Server-side** The 'side' of the communication where the server is. In browsing The Internet it's the side of the computers hosting the content that is being requested.

**Silk road** Online drugs and weapon marketplace in TOR lead by the arrested Ross William Ulbricht on which an estimated \$15 million per year was spend on mostly illegal contraband. More on the Silk road: <http://securityaffairs.co/wordpress/8005/cyber-crime/traveling-the-silk-road-study-of-the-famous-market-places.html>.

**Single point of failure** The entity in a system where the entire system relies upon. Taking this entity down will take the entire system down.

**Single point of trust** The entity in a system which is trusted by everybody in the system. When this entity is hacked or malicious, the entire cryptographic system is not to be trusted.

**SSL certificate** File containing a public key of a key pair of which only the company mentioned in the certificate has the private key of. Certificates are issued by certificate authorities. Certificates not issued by official certified certificate authorities are not trusted by default in web browsers.

**Symmetric Encryption** Mathematical transformation of a value to another value using a key. Decryption with the same key leads to the same original value. Symmetric encryption is mostly based on the infeasibility of deriving shared prime factors.

**Tarzan** Attempt to anonymize the Internet. See section [2.3](#).

**Tit-for-tat** Mechanism used in BitTorrent which gives users that upload much higher download speed.

**TOR** The Second Generation Onion Router. Uses multi hop onion routing to ensure anonymity. TOR is one of the most used tools for online anonymity and is mainly used in the TOR browser which can browse the world wide web. Available from the TOR website <http://torproject.org>.

**Tracker** Server that keeps track of users that are downloading and uploading parts of a torrent file.

**Tribler** Peer to peer file sharing tool developed at the Delft University of Technology by the department of Parallel and Distributed Systems, Delft. Available for download for free at <http://tribler.org>.

**XKeyScore** Program of the NSA that combines all information sources. Edward Snowden disclosed that with XKeyScore any computer, cellphone and email are trackable and compromised with XKeyScore. More information in a german interview with Edward Snowden: <http://www.commondreams.org/headline/2014/01/27-1>.

# Acronyms

**AES** Advanced Encryption Standard.

**CBC** Cipher Block Chaining.

**CTR** Counter Feedback.

**DAS4** Distributed ASCII Supercomputer 4th generation.

**DHT** Distributed Hash Table.

**DTLS** Datagram Transport Layer Security.

**EC** Elliptic Curve.

**ECB** Electronic Code Book.

**HTTP** HyperText Transfer Protocol.

**HTTPS** HyperText Transfer Protocol Secure.

**I2P** Invisible Internet Protocol.

**ISP** Internet Service Provider.

**KLOC** 1000 Lines Of Code.

**MITM** Man In The Middle.

**NAT** Network Address Translation.

**NSA** National Security Agency.

**OR** Onion Routing.

**P2P** Peer to Peer.

**PFS** Perfect Forward Secrecy.

**RSA** Rivest, Shamir & Adleman.

**TCP** Transmission Control Protocol.

**TOR** The Onion Router.

**μTP** Micro Transport Protocol.

**UDP** User Datagram Protocol.



# References

- [1] T. Berners-Lee, R. T. Fielding, and H. F. Nielsen, *RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0*, <http://www.faqs.org/rfcs/rfc1945.html> (1996).
- [2] E. Rescorla, *HTTP Over TLS*, RFC 2818 (Informational) (2000).
- [3] R. Dingledine, N. Mathewson, and P. Syverson, *Tor: The second-generation onion router*, in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04 (USENIX Association, Berkeley, CA, USA, 2004) pp. 21–21.
- [4] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21**, 120 (1978).
- [5] B. Kaliski, *PKCS #1: RSA Encryption Version 1.5*, RFC 2313 (Informational) (1998), obsoleted by RFC 2437.
- [6] D. Eastlake 3rd and P. Jones, *US Secure Hash Algorithm 1 (SHA1)*, RFC 3174 (Informational) (2001), updated by RFC 4634.
- [7] D. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, *High-speed high-security signatures*, *Journal of Cryptographic Engineering* **2**, 77 (2012).
- [8] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard* (Springer Verlag, Berlin, Heidelberg, New York, 2002).
- [9] W. Diffie and M. Hellman, *New directions in cryptography*, *IEEE Trans. Inf. Theor.* **22**, 644 (2006).
- [10] D. Harkins and D. Carrel, *The Internet Key Exchange (IKE)*, RFC 2409 (Proposed Standard) (1998), obsoleted by RFC 4306, updated by RFC 4109.
- [11] R. Dingledine and S. Murdoch, *Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it*, Tech. Rep. (The Tor Project, 2009).
- [12] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, *Privacy-preserving p2p data sharing with oneswarm*, in *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10 (ACM, New York, NY, USA, 2010) pp. 111–122.

- [13] S. Prusty, B. N. Levine, and M. Liberatore, *Forensic investigation of the oneswarm anonymous filesharing system*, in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11 (ACM, New York, NY, USA, 2011) pp. 201–214.
- [14] M. J. Freedman and R. Morris, *Tarzan: A peer-to-peer anonymizing network layer*, in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02 (ACM, New York, NY, USA, 2002) pp. 193–206.
- [15] B. Schneier, *Description of a new variable-length key, 64-bit block cipher (blowfish)*, in *Fast Software Encryption, Cambridge Security Workshop* (Springer-Verlag, London, UK, UK, 1994) pp. 191–204.
- [16] M. O. Rabin, *DIGITALIZED SIGNATURES AND PUBLIC-KEY FUNCTIONS AS INTRACTABLE AS FACTORIZATION*, Tech. Rep. (Cambridge, MA, USA, 1979).
- [17] D. Fifield, N. Hardison, J. Ellithorpe, E. Stark, D. Boneh, R. Dingledine, and P. Porras, *Evading censorship with browser-based proxies*. in [33], pp. 239–258.
- [18] T. E. Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31**, 469 (1985).
- [19] T. Pornin, *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*, RFC6979 (2013).
- [20] T. Hansen, *US Secure Hash Algorithms*, RFC6234 (2011).
- [21] S. Farrell, T. C. Dublin, and T. Tschofenig, *Pervasive monitoring is an attack*, <https://www.rfc-editor.org/rfc/rfc7258.txt> (2014).
- [22] B. Cohen, *Incentives build robustness in bittorrent*, (2003).
- [23] N. Christin, *Traveling the silk road: A measurement analysis of a large anonymous online marketplace*, CoRR **abs/1207.7139** (2012).
- [24] G. Halkes and J. Pouwelse, *Udp nat and firewall puncturing in the wild*, in *Proceedings of the 10th International IFIP TC 6 Conference on Networking - Volume Part II*, NETWORKING'11 (Springer-Verlag, Berlin, Heidelberg, 2011) pp. 1–12.
- [25] D. Genkin, A. Shamir, and E. Tromer, *Rsa key extraction via low-bandwidth acoustic cryptanalysis*, IACR Cryptology ePrint Archive **2013**, 857 (2013).
- [26] V. S. Miller, *Use of elliptic curves in cryptography*, in *Lecture Notes in Computer Sciences; 218 on Advances in cryptology—CRYPTO 85* (Springer-Verlag New York, Inc., New York, NY, USA, 1986) pp. 417–426.
- [27] N. Zeilemaker, B. Schoon, and J. Pouwelse, *Dispersy Bundle Synchronization*, Tech. Rep. PDS-2013-002 (TU Delft, 2013).

- [28] T. Kivinen and M. Kojo, *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*, RFC 3526 (Proposed Standard) (2003).
- [29] R. Tanaskoski, *Anonymous HD Video Streaming*, Master's thesis, Delft University of Technology, Delft, the Netherlands (2014).
- [30] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips, *Tribler: A social-based peer-to-peer system: Research articles*, *Concurr. Comput. : Pract. Exper.* **20**, 127 (2008).
- [31] A. Friedl, S. Ubik, A. Kapravelos, M. Polychronakis, and E. Markatos, *Realistic passive packet loss measurement for high-speed networks*, in *Traffic Monitoring and Analysis*, Lecture Notes in Computer Science, Vol. 5537, edited by M. Papadopouli, P. Owezarski, and A. Pras (Springer Berlin Heidelberg, 2009) pp. 1–7.
- [32] M. Meulpolder, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, *Bartercast: A practical approach to prevent lazy freeriding in p2p networks*. in *IPDPS* (IEEE, 2009) pp. 1–8.
- [33] S. Fischer-Hübner and M. Wright, eds., *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, Vol. 7384 (Springer, 2012).



