FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Lower-Latency Anonymity

Latency Reduction in the Tor Network using Circuit-Level Round-Trip-Time Measurements

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

by

Robert Annessi
Registration Number 0527718

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Privatdoz. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Edgar Weippl
Assistance: Univ.Lektor Dipl.-Ing. Dr.techn. Martin Mulazzani

August 2014

Acknowledgements

I express my very special thanks to:

Mike Perry
Matthias Fassl
Martin Mulazzani
Caroline Durlacher
David Kaufmann
Martin Sereinig
Jakob Bleier
Damian Johnson
Paul Syverson
Anita Schraffl
Franziska Annessi

Furthermore, I want to explicitly thank Roger Dingledine, Nick Mathewson, Aaron Johnson, Gabriele Eva Maria Höfler, and Edgar Weippl.

Thanks also go to the following organizations:

[1] https://bitbucket.org/ra_/tor-rtt/
[2] http://128.130.204.91/ra-torlatency-data.tar

Abstract

With the tremendous increases in communication over the Internet, privacy issues have become more and more important. In the interest of allowing people to communicate without revealing potentially identifying information, much research and effort has been put forth to develop anonymous communication protocols, which became the technical basis for promoting freedom of speech, achieving privacy, and overcoming censorship on the Internet. The most widespread and well researched anonymity system is Tor, which achieves a reasonable balance between the conflicting demands of performance and security.

Although both latency and throughput have been improved significantly in recent years, Tor users still occasionally experience long and variable delays. Such delays are not only harmful for interactive web users, who create the vast majority of connections in the Tor network, but they also prevent altogether the use of real-time protocols, such as the Voice-over-Internet Protocol, where a certain quality of service is indispensable.

In this thesis we find our means to decrease latency, the most important property from users' perspective. In our approach, clients actively measure Round-Trip-Times (RTT) of circuits after they have been established and drop slow circuits before they begin to be used. We conduct several experiments on the live Tor network, to verify our assumption that the use of lightweight, active RTT measurements can achieve latency improvements. Our results show that this approach achieves an improvement not only in latency, but also in throughput, and in anonymity.

Keywords: Anonymity; Tor; Latency

Contents

1 Introduction **1**
1.1 Anonymous Communication 2
1.2 Thesis 4

2 Background **7**
2.1 Tor's Architecture 8
2.2 SOCKS 12
2.3 Security Challenges 13

3 State of the Art **15**
3.1 Guard Nodes 15
3.2 Path Selection 17
3.3 Congestion Avoidance 20
3.4 Tor Control Protocol 21

4 Related Work **23**
4.1 RTT-Measurements 23
4.2 Performance 25
4.3 Network Adversaries 28
4.4 Summary and Analysis of Existing Approaches 32

5 Methodical Approach **35**
5.1 Used Concepts 36
5.2 Path Generator 38
5.3 Design Methods 40
5.4 Experimental Environment 42
5.5 Measurement and Analysis Methods 44

6 Results **47**
6.1 RTT Measurements on Individual Circuits 47
6.2 Single RTT Measurements on Various Circuits 49
6.3 Measuring RTTs, CBTs, and TTFBs on Various Circuits 51
6.4 Calculating Cut-Off Values 53
6.5 Latency Evaluation 56

6.6 Bandwidth Evaluation 59
6.7 Anonymity Evaluation 61
6.8 Guard Nodes 63

7 Discussion **67**
7.1 Limitations 67
7.2 Future Work 69

8 Conclusion **71**
8.1 Contributions 71
8.2 Suggested Implementations 73

Bibliography **75**

Acronyms **81**

CHAPTER 1

Introduction

> *Man is least himself when he talks in his own person. Give him a mask, and he will tell you the truth.*

Oscar Wilde, *The Critic as Artist*, 1891

A fundamental building block of today's Internet is the Internet Protocol (IP) [1], published in 1974. It facilitated the delivery of data packets between host computers regardless of the underlying telecommunication networks. To this end, the Internet Protocol introduced an addressing scheme that assigns a globally unique[1] identifier to every computer on the Internet, the so-called *IP address*. While such a globally unique identifier for every computer made routing packets at the network level a relatively easy task, at the same time it poses an inherent problem to preserving Internet users' privacy. When a packet is forwarded on a communication path, any intermediate router can observe the addresses of both the sender and the recipient, possibly inferring information about the behavior of users. It is important to understand that the security and privacy of communicating parties went completely unconsidered in the initial specification of the Internet Protocol. Its primary objective was to establish an protocol enabling computers on different telecommunication networks to communicate on a global scale.

[1] In fact, not every computer on the Internet is assigned a globally unique IP address anymore. In 1996, an addressing scheme for private networks using IP addresses that are not globally delegated was introduced by Request for Comments (RFC) 1918 [2]. If a host computer on such a private network wants to establish a connection to the Internet, it has to use a Network Address Translation (NAT) gateway that maps several private IP addresses to a single public IP address so that they appear to originate from the same gateway. In general, NAT gateway operators do know who used their gateway at a certain point in time, again making computers identifiable even though they do not have an assigned public IP address.

Since then, with the tremendous increases in communication over the Internet, security and privacy issues have become more and more important. Therefore, new protocols for protecting the confidentiality, authenticity, and integrity of communications have been developed to protect sensitive communications such as personal emails or financial transactions. While such protocols hide the contents of communications from unauthorized third parties, they cannot conceal the fact that two specific parties are communicating at all.

1.1 Anonymous Communication

In the interest of allowing people to communicate without revealing potentially identifying information, such as their computers' IP addresses, much research and effort has been put forth to develop anonymous communication protocols. Without redesigning the fundamental architecture of the Internet, such protocols seek to enable people to communicate while concealing the identities of those communicating with one another. Hiding relationships between communicating parties, anonymous communication systems have become the technical basis for promoting freedom of speech, achieving privacy, and overcoming censorship on the Internet.

Background

Chaum's untraceable email [3] in 1981 first introduced the notion of modern anonymous communication systems. In order to allow an electronic mail system to conceal with whom a user communicates, he proposed sending messages through a set of servers. Each of these servers would shuffle messages from several senders before forwarding the messages to their next destinations, actively hiding the relationships between senders and receivers. Furthermore, by using public key cryptography even the content of the communication remains private, in spite of the underlying unprotected telecommunication networks. In order to determine both the sender and the recipient of a message, it would be necessary for all servers involved to collude. In other words, only a single benevolent server in a set is required to successfully anonymize the communication.

Since then, a variety of other anonymous communication systems have been proposed. Yet all rely to some extent on this idea of forwarding messages through anonymizing servers. The most fundamental techniques for realizing anonymous communication on the Internet can be divided in two categories: *high-latency* and *low-latency* anonymity systems.

High-Latency Anonymity Systems

The first anonymity systems actually deployed on the Internet were high-latency anonymity systems like Babel [4], Mixminion [5], and Mixmaster [6], which are message-oriented and largely based on the principles proposed by Chaum.

With the goal of maximizing anonymity, high-latency systems deliver messages after a significant delay, around four hours on average; the effective delay of message delivery can even amount to as much as several days. Additionally, servers reorder messages before forwarding

them in batch to their intended destinations. Hiding real messages among random noise, continuously generated random cover traffic helps to defend against traffic analysis, which can be used to identify a sender. Some high-latency systems use such cover traffic methods, despite its vast use of additional networking resources.

With these large and variable delays, high-latency networks are able to a very large degree impede strong adversaries from inferring the sender and recipient of a message, even if an adversary is assumed to be capable of monitoring all communication links globally. In the literature, such powerful attackers are called *global passive adversaries*.

Although high-latency anonymity techniques provide very high levels of security by attempting to conceal timing information and thereby frustrate traffic analysis, they are still vulnerable under certain conditions: if a user's habits are sufficiently distinct and his or her message volume is high enough. [see 7, 8]

Because their optimization for anonymity entails delays of variable length, high-latency systems are in practice usable only for non-interactive applications, such as email communication. They simply produce too much lag for interactive tasks like web browsing, Internet chat, or Secure Shell (SSH) connections.

Low-Latency Anonymity Systems

In contrast to high-latency systems, which attempt to distort timing patterns, low-latency systems like Crowds [9], Freedom [10], and The Onion Router (Tor) [11] are able to provide higher speed by actively seeking to limit processing delay and bandwidth overhead. This allows low-latency anonymity systems to facilitate anonymous use of interactive real-time applications like web browsing, instant messaging or SSH connections. However, the increased speed comes at the cost of compromised anonymity. There exists an inherent trade-off between performance[2] and security: Low-latency anonymity systems are far less resistant to traffic analysis by a global passive adversary capable of monitoring all communication links. Such an adversary can analyze the timing and volume of traffic entering and leaving the anonymity network, and thus deduce which parties are communicating with one another and successfully deanonymize users. [cf. 12]

Despite the lower theoretical limit to the anonymity such systems can achieve, it is still possible in practice for users of low-latency systems to reach a higher degree of anonymity, then can be reached by users of high-latency anonymity systems. Though this may be counter-intuitive at first glance, it should become clear, in the context of the common definition of anonymity:

> *"Anonymity of a subject means that the subject is not identifiable within a set of subjects, the anonymity set."* [13, p. 9]

Following to this definition, to an attacker an anonymous subject is indistinguishable from other subjects within the anonymity set. [see 13, p. 9] Because anonymity systems hide users among users, the degree of anonymity for each person using an anonymity system is closely tied to the size of the anonymity set. Therefore, although low-latency systems lack the exceptional anonymity properties of high-latency systems, they can still provide greater anonymity in practice, as long as the anonymity set is sufficiently large.

[2] In terms of latency and throughput.

The Onion Router

Among the very few systems that are of practical relevance for providing anonymity is Tor, short for *The Onion Router*, which is loosely based on the onion routing [14] principle. In operation since 2003, the Tor network first gained public attention in 2004. Since then, it has become with millions of daily users [see 15] from over 120 different countries [see 16, p. 51] the most widespread low-latency anonymity network and the platform for research in anonymous communications. Like other low-latency anonymity designs, Tor seeks to hide the relationships between communicating parties both from network observers and from the anonymization infrastructure itself. At the same time, Tor developers strive to achieve a reasonable balance between the conflicting demands of performance and security in order to enable the use of interactive real-time applications, such as web browsing, and thus encourage use of the network in general.

1.2 Thesis

Problem Statement

Since Tor intentionally bounces traffic around the world several times, the best achievable latency when using Tor is by design higher than on regular Internet connections. Although both end-to-end latency and throughput in the Tor network have been improved significantly over the last years [see 16, pp. 67-70], users still often experience long delays [cf. 17, 18, 19]. This is especially harmful for real-time applications sensitive to latency, such as interactive web traffic, which represents the vast majority of connections [see 16, p. 50, 20] through the Tor network.

There are several reasons why latencies are often unacceptably high for delay-sensitive web users. First, the bulk data transfers of file-sharing users consume a vastly disproportionate amount of the network's bandwidth, degrading the performance of the Tor network. [see 16, p. 78] Furthermore, some nodes are highly congested, since the Tor network does not have enough capacity available [cf. 17] for its huge number of users.

These end-to-end delays are not only often perceived as unacceptably high by delay-sensitive interactive web users, but they also prevent altogether the use of real-time applications like Voice over IP (VoIP) where a certain quality of service is indispensable.

Aim of the Work

In this thesis, we seek to improve the performance and security properties of the Tor network by reducing latency such that real-time applications benefit from faster connections and users are less frustrated. Furthermore, our method aims to enable the anonymous use of applications like VoIP, which could not previously be anonymized.

We argue that keeping the performance costs associated with the anonymity system are as low as possible would make using Tor more attractive both to existing users, who would use Tor more regularly, and to potential users, who might join the network. Because the degree of anonymity provided by such system is highly related to the number of its users, enlarging the user base ultimately enhances the system's anonymity properties, rather than compromising them.

Methodological Approach

In order to reduce latency in the Tor network, we actively measure the time it takes for data packets to go through the Tor network and back, the so called Round-Trip-Time (RTT). This active measurement approach is based on a technique originally proposed by Panchenko and Renner [21] in 2009, later also employed by Wang et al. [19], and Panchenko, Lanze, and Engel [22] in 2012.

User experienced network latency is approximated with the time elapsed between sending a Hypertext Transfer Protocol (HTTP) request to the "google.com" website through the Tor network and receiving the first byte of its response. Achieved bandwidth over Tor circuits is measured by the time it takes to download a 5 MB static HyperText Markup Language (HTML) file from Cloudflare's Content Delivery Network (CDN) through the Tor network. Two different methods are used to quantify anonymity:

- Normalized Shannon-Entropy over the distribution of entry and exit node combinations, as proposed by Serjantov and Danezis [23], and Diaz et al. [24]
- Inequality in node selection through an adopted Gini coefficient [25], as proposed by Snader and Borisov [26].

We will show that our lightweight, active measurement approach can not only be used to avoid high-latency circuits, but also improve bandwidth and anonymity! Furthermore, we will explore whether extensive active measurements, despite lowering the degree of anonymity, can be used to assure the quality of service required by real-time applications like VoIP. Clients can implement the proposed algorithm themselves without requiring a network-wide upgrade. Furthermore, our method increases the load neither on the clients nor on the Tor network significantly. To validate our approach, we conduct experiments on the live Tor network operating from PlanetLab [27] hosts on multiple networks.

Structure of the Work

The remainder of this thesis is organized as follows: Following this Chapter we give an overview of Tor's technical basis covering such concepts as directory authorities, path selection, exit nodes, circuit building and cells. Moreover, we explain how user applications communicate through Tor and outline known security challenges. In the third Chapter, we describe state-of-the-art methods used in Tor, such as Guard nodes, details on path selection, congestion related methods, and the Tor Control Protocol, all of which are important to understand our approach. Next, we present a summary of related works that seek to find improvements to Tor's performance and anonymity in Chapter 4. In addition to describing our own approach, we specify in Chapter 5 the methods we use to measure latency and anonymity, and our experimental setup. We analyze and evaluate the results of our experiments in Chapter 6. Limitations of our evaluation and an outlook, discussing possible future work, are provided in Chapter 7. Finally, we conclude in Chapter 8.

CHAPTER 2

Background

> *The fascinating technological problems and the potential to better protect people and their activities was nice, but the real attraction for creating onion routing was to create a context where people, who were sure they should hate each other, were forced to collaborate.*

Paul Syverson, *In an "only somewhat facetious" conversation at the Financial Cryptography Conference*, 1998

Presently, only very few systems actually exist that are of practical relevance for providing anonymity on the Internet. The most widespread and well researched is Tor [11], short for *The Onion Router*, a low-latency anonymity system, loosely based on the onion routing [14] principle. In operation since 2003, the Tor network gained first public notice in 2004. Since then, it has become the most popular low-latency anonymity network with millions of daily users [see 15] from over 120 different countries [see 16, p. 51] and the platform for research in anonymous communications. Available under the free and open source Berkeley Software Distribution (BSD) license, Tor has changed since its initial design significantly through an active and open volunteer-driven development process.

Like other low-latency anonymity designs, it seeks to hide the relationships between communicating parties both from any network observers and from the anonymization infrastructure itself. At the same time, Tor developers strive to achieve a reasonable balance between performance and security in order to encourage the use of the network. Since Tor achieves relatively low latency and high throughput, it is ideal for interactive applications such as web browsing or instant messaging.

As the most successful and widely adopted anonymity solution on the Internet, Tor is used daily for a wide variety of purposes by professionals such as journalists, researchers, and human rights activists, organizations such as businesses, militaries, and law enforcement, privacy aware people, and many others. A growing number of people require not only anonymous communications but also communications resistant to censorship, particularly in those countries where the policies of local governments restrict Internet freedoms. [see 28, 16, p. 61]

Perhaps one of the greatest explanations for Tor's success is that virtually any application can be anonymized through the Tor network, as long as the application supports the Transmission Control Protocol (TCP). Furthermore, Tor uses the standard Socket Secure (SOCKS) [29] proxy interface which many TCP-based applications support by default. [see 28] In practice, when a user's application like a web browser or email client wants to establish a connection, it asks the Tor client software via SOCKS to make that connection.

Structure The remainder of this Chapter is organized as follows: In the next Section (2.1) we give an overview of Tor's architecture, describing Tor's basics and fundamentals in detail, such as directory authorities, path selection, exit nodes, circuit building and cells. Finally, we describe the SOCKS proxy interface and close with known security challenges in Tor.

2.1 Tor's Architecture

Tor is a circuit-switched, low-latency and bandwidth-efficient anonymizing communications network. It is designed to provide source and destination anonymity for applications built on TCP, such as web browsing, secure shell, and instant messaging. Tor's system architecture is illustrated in Figure 2.1 on the facing page. At the time of writing, in early 2014, the publicly accessible Tor network consisted of a few semi-centralized directory servers and about 5000 volunteer-operated nodes relaying more than 40 Gbit/s of traffic. [see 30]

Users run the *Tor client software*[1] as a normal user-level process without any special privileges on their computers. This client software provides a SOCKS-proxy interface on the user's computer to relay the data of incoming TCP connections as streams through the Tor network. [see 28]

To establish an anonymous communication channel to the user's intended destination, the Tor client software has to choose a path through the Tor network first. When relaying traffic on behalf of users, each Tor node along a path rewrites the IP packet's source and destination addresses in such a way that hides the true sender and receiver. A key aspect of achieving anonymity is that each node in a path knows only its predecessor and successor, but no single node knows the identities of both communicating parties. Encrypting data packets multiple times before sending them through an anonymization path ensures that no one but the user knows the true identity of both sender and receiver - not even an eavesdropper monitoring the user's link. Thus, the first node knows the identity of the user contacting it, in the form of its IP address, but not that of the destination. The last node knows the destination of the user's communication, but not the user's identity.

[1] often referred to as *Onion Proxy (OP)*

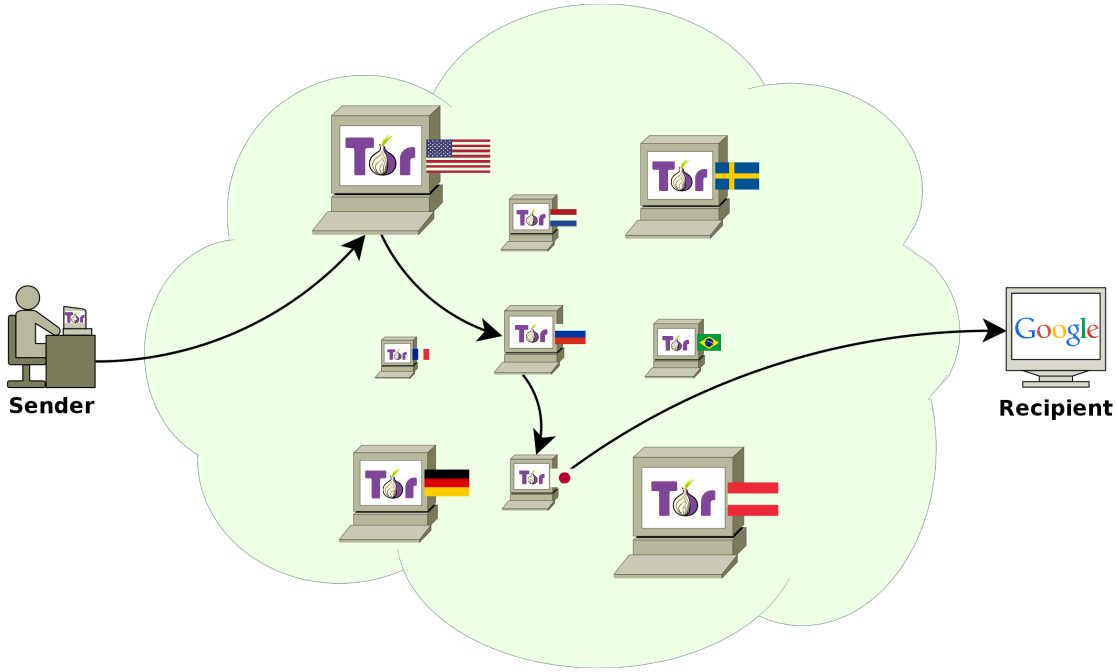

Figure 2.1: Tor's system architecture

Directory Authority

Before selecting a path through the Tor network, the client software needs to obtain a list of available nodes. A small group of redundant, trusted servers act as the *directory authority* which keeps track of the status of nodes in the Tor network. The directory authority provides fundamental information on recognized nodes in the network, commonly referred to as the *directory consensus*. Along with other information, the consensus document contains the nodes' IP addresses, their public encryption keys and information regarding their capacities.

In addition, the directory consensus document contains so called *flags* for each node, like the *stable* and *fast* flags, for example. Persistent applications, such as the File Transfer Protocol (FTP), SSH or instant messaging, which establish continuous sessions, require more stable connections than applications with short-lived sessions like web browsing. For such long-lived applications, the Tor client software chooses anonymization paths which solely consist of stable nodes. A node is marked as stable by the directory authority, if its has been continuously running for at least 30 days or its uptime value is above the median for known active nodes. In terms of bandwidth, the most powerful 87.5% of nodes are marked as *fast*. [cf. 31]

To update its view of the network, the Tor client software periodically downloads the consensus document from a directory authority tunneled through an anonymized Tor connection. Some nodes choose to act as directory cache, so that clients can contact them to avoid excessive load on the directory authority. The directory authorities combine their own views of the composition of the Tor network and status of all nodes. They periodically vote to determine a directory consensus of the entire network state. [see 28]

As soon as enough information from a directory authority is gathered, the Tor client software can begin selecting *paths* through the Tor network.

Path Length

To establish an anonymous communication channel to the user's intended destination, the Tor client software first has to select a path through the Tor network by choosing suitable nodes among all active ones.

If a path consisted of a single node only, then this particular node would trivially know both the user's and the destination's identities. Since anyone can join the Tor network by running a Tor node and thus offer available resources to other users, it is reasonable to assume that an adversary is capable of controlling a single node. Therefore, the minimum length for general purpose paths in Tor is set to two. Nevertheless, like any other low-latency anonymity system, Tor is vulnerable to end-to-end timing attacks, where both the first and the last hop of a path collude. With a path length of two, it would be easily possible for both nodes involved to deanonymize sender and recipient, under the assumption that the nodes in the path colluded. To mitigate this threat, the number of nodes used in an anonymization path must be sufficiently high. But even if a client used a very long path, an adversary capable of performing complex, non-trivial timing analysis could still learn the identities of both communicating parties, provided that the first and the last node of a path are controlled by the adversary. However, performance degrades as path length increases, since throughput decreases and latency increases with every additional node. [see 32] Therefore, the default path length of three nodes finds a suitable balance in trade-off between security and performance.

The first, second, and third node of a path are called *entry*, *middle*, respectively *exit* nodes. Tor's path selection procedure will be presented in greater detail in Section 3.2, which begins on page 17.

Exit Nodes

The last node of an anonymization path is called exit node. Providing a gateway between the Tor network and the potentially non-encrypted Internet, the exit node is responsible for establishing the connection to the user's intended destination. Since it forwards traffic on behalf of Tor users, the exit node's IP address appears to be the source of communication for the recipient. Thus, if the destination detects any malicious activity, such as hacking attempts or copyright infringement, it will assume that the exit node is responsible. Such abuse can lead to legal prosecution in some countries, limiting the number of operators willing to run Tor exit nodes.

To reduce the possibilities for abuse, node operators can specify so-called *exit policies* to narrow or prohibit connections to hosts on the Internet. Each node's exit policy specifies to which IP addresses and TCP ports the node is willing to relay traffic. A mixture of open and restricted exit nodes allows the most flexibility for volunteers running Tor nodes, without necessarily having to worry about abuse issues. However, abuse of exit nodes remains only incompletely prevented. [see 28]

Thus, when selecting an exit node, clients are constrained not only by these policies, which determine whether a node is willing to serve as an exit for a particular destination, but also by

the sheer scarcity of nodes configured to be used as exit nodes at all - only about one third of all nodes.

Circuit Building

As soon as the Tor client software has successfully obtained the list of Tor nodes, including the nodes' public encryption keys from a directory authority, and selected a path, the Tor client software iteratively negotiates encryption keys with each of the nodes involved. The resulting encrypted connection through the Tor network is called a *circuit*.

The Tor client software creates circuits in such a way that negotiates session keys between the client and each node on a path in a telescoping fashion: The Tor client software constructs circuits incrementally, one hop at a time, using each partially created circuit to communicate with the next node. [see 28, 16, p. 27]

First, in order to establish an encrypted tunnel with the entry node, the Tor client software uses Diffie-Hellman key exchange to negotiate a shared symmetric session key. Subsequently, to extend the encrypted tunnel further to the middle node, the Tor client software connects to it through the previously established encrypted tunnel with the entry node and negotiates another shared symmetric session key. Finally, another session key is negotiated with the last node through the newly established encrypted tunnel, completing the bidirectional, real-time virtual circuit of layered encryption.

Once the circuit is closed, these session keys are deleted so that subsequently compromised nodes cannot decrypt old traffic, achieving perfect forward secrecy and key freshness. In addition, this circuit-level handshake protocol results in unilateral entity authentication, as the Tor client software can verify the identity of each node in a circuit, but the client remains anonymous, having used no public key. [see 28]

Preemption Constructing a circuit involves computationally expensive public-key cryptography and multiple packet round trip times, which typically takes up to several seconds. To avoid propagating such high delay to the user, the Tor client software automatically attempts to maintain several preemptively built circuits. Circuit building time thus does not harm user experience as long as a suitable circuit is available when an application makes a request.

Dirty Circuits Since significant cryptographic overhead is involved in their creation, circuits are reused for multiple TCP streams to improve efficiency. However, to prevent users from certain profiling attacks, the Tor client software avoids re-using a specific circuit, if the circuit's first stream was created more than ten minutes ago.[2] Nonetheless, once a TCP connection has been established over a circuit, it remains on that circuit until it is closed, even beyond that time limit. [see 28]

Once a circuit has been established, the client shares symmetric keys with each node along the circuit and can begin transmitting packets.

[2]The default value for *"MaxCircuitDirtiness"*.

Cells

To mitigate traffic analysis attacks that try to correlate packets entering and leaving the Tor network based on the packets' sizes, all packets are transmitted in fixed-size 512-byte units called *cells*. Each cell consists of a header that includes a circuit identifier and a payload that is padded when not enough data is available. Requiring only symmetric key encryption, relaying cells is computationally inexpensive compared to building circuits, which involves public-key cryptography. [see 28]

Before forwarding cells, the client encrypts them appropriately: Initially, the cells are encrypted using the key shared with the exit node, then again using the key shared with the middle node, and finally using the key shared with the entry node. The cells are then forwarded to the entry node, which removes the outermost layer of encryption, resulting in data encrypted to the middle node. Note that the entry node cannot read the ultimately resulting cell's payload, as it does not know the key shared between the client and the middle node. The entry node then passes the cells on to the middle node, which decrypts another layer and forwards the cells to the exit node. The exit node removes the last encryption layer, revealing the final destination's IP address and the payload, such as a request for a website.

In this way each of these three nodes only knows the identity of the node before and after it on the path, and only the exit node learns the final destination of the packet. That is, the entry node knows the identity of the client and of the middle node, the middle node knows the identity of the entry and exit nodes, and the exit node knows the identity of the middle node and the actual destination of the packet. The response packet is then relayed back along the same three nodes in the opposite direction.

2.2 SOCKS

The initial implementation of onion routing required a distinct proxy for each application protocol - most of which were never written, so that many applications were never supported. On the other hand, Tor uses the standard SOCKS[3] proxy interface, which many TCP-based applications support without requiring any modification. [see 28]

In practice, when a user's application like a web browser or email client wants to establish a TCP connection to a given IP address (or domain name) and TCP port, it asks the Tor client software via SOCKS to make that connection. SOCKS routes network packets between the user's application and the Tor client software, acting as transparently as possible. The Tor client software then chooses a suitable, existing circuit or, if needed, creates one, and accepts data from the application's TCP stream, packaging it into cells and sending those cells along the encrypted circuit. [see 28]

However, if the user's application does Domain Name System (DNS) resolution first, rather than sending the host name through the Tor network to be resolved by the exit node, it reveals its destination to the remote name server. To avoid this vulnerability, it is important that the user

[3]SOCKS is an Internet protocol that performs at the fifth layer of the seven-layer Open Systems Interconnection (OSI) model. In the OSI model of computer networking layer five is the session layer.

configures applications to use SOCKS version 4A or 5, so that applications send host names rather than IP addresses. [see 28]

2.3 Security Challenges

Protocol normalization The Tor client software itself does not provide any protocol normalization. For example, the protocol cleaning of HTTP and the Hypertext Transfer Protocol Secure (HTTPS) relies on patching and careful configuration of the Firefox web browser, to restrict tracking capabilities such as cookies, to normalize browser characteristics accessible from JavaScript such as screen size and system colors, and to block plug-ins which may leak identifying information. [see 28]

Tor does not protect against application-level attacks that reveal the user's IP address. For a privacy-enhanced browser, Tor relies on the Tor Browser Bundle (TBB) to incorporate modifications made to the Firefox web browser. [cf. 33]

Linking Attacks Tor allows multiple TCP streams to share a single circuit, which improves efficiency but also facilitates so-called linking attacks. Protocols that commonly leak identifying information, such as HTTP or instant messaging, can be linked to the same originating user by an exit node's operator or an adversary capable of observing an exit node's traffic. For example, a user accessing multiple pseudonymous chat accounts simultaneously would reveal to the exit node that the accounts were shared by the same user. To avoid such linking attacks, the Tor client software can be configured to isolate streams based on certain properties, like the user's application or the destination host, so that, for example, the user's anonymous web browsing never shares a circuit with her pseudonymous instant messaging usage. [see 28]

Passive Observation Since, in general, data is transferred unencrypted over the link from the exit node to the destination host, and insecure protocols transmitting passwords in plain text are still fairly common, malicious exit nodes could potentially gather passwords simply by observing traffic. [see 16, p. 50] Therefore, it is highly recommended that users configure their applications explicitly to use additional encryption.

End-To-End Timing Correlation Tor only minimally hides end-to-end timing correlations, leaving it vulnerable to *correlation attacks*. Assuming that both the first and the last hop of a path collude, or an adversary is capable of observing the traffic at both the sender and the receiver, the correspondence can be confirmed with high probability. Currently, users can protect themselves best against such an attack by running a Tor node on their own host computers. [see 28] Section 3.1 on page 17 discusses end-to-end timing correlation attacks in greater detail.

CHAPTER 3

State of the Art

> *It only takes 20 years for a liberal to become a conservative without changing a single idea. In an evolving universe, who stands still moves backward.*

Robert Anton Wilson, *The Illuminati Papers*, 1980

Available under the free and open source BSD license, Tor's initial design has changed significantly through an active and open volunteer-driven development process. In this Chapter we will describe the state-of-the-art concepts important to understanding our approach.

First, we explain the concept of Guard nodes and the rationale behind them, starting on the current page. Tor's path selection algorithm is presented in detail in the subsequent Section on pages 17–19. Afterwards, two methods aiming to avoid congestion in the Tor network are explained in Section 3.3 on page 20. Finally, in the last Section on page 21, we close this Chapter by briefly describing the Tor Control Protocol.

3.1 Guard Nodes

The first nodes in Tor's anonymization paths are called *entry* nodes. Instead of choosing a new entry node for each path, every Tor client[1] restricts its choice of potential entry nodes to a small semi-persistent set. This concept of *helper nodes* was originally invented by Wright et al. [34] in 2003 and later proposed for use in Tor as *Guard nodes* by Øverlier and Syverson [35] in 2006. [see 28]

[1] Beginning with this Chapter, we denote to the Tor client software simply as "client".

Guard nodes were introduced to mitigate both the efficiency of the predecessor attack [36] and the threat of statistical profiling, in which an adversary observes users' traffic entering and exiting the anonymity network (e.g. by controlling or monitoring the first and last node of circuits). Both types of attacks will be explained in detail, beginning on this page.

Clients select their set of entry Guards from all nodes that have the *Guard* flag assigned by the directory authority, which are roughly one third of all Tor nodes. Whenever it selects a path, the client picks an entry node among its selection of Guard nodes. By default, Tor clients maintain a list of precisely three entry Guards.

The Guard flag is assigned to nodes which are known for at least one week and whose bandwidth is at least 250 kB/s or above the median bandwidth of all nodes.[2] In addition, a node's weighted fractional uptime[3] has to be above the median of all nodes before it is assigned the Guard flag. To calculate weighted fractional uptime, the directory authority computes the fraction of time that the node is up in any given day, weighting so that downtime and uptime in the past counts less. [see 31]

The expiration time, after which clients select new Guard nodes, was increased from 30 to 60 days in Tor version 0.2.4.12-alpha. In the future, the size of the set of Guard nodes may be reduced further [see 37] and their expiration time increased, to improve security as proposed by Elahi et al. [38] and confirmed by Johnson et al. [39]. A summary of related works regarding Guard nodes will be presented in Section 4.4 on page 34.

Predecessor Attack

As entry to the Tor network, the first node on a path directly learns the IP address of the sender, which could potentially identify the user. Given Tor's volunteer-driven network, it is reasonable to assume that an adversary is capable of running nodes and thus can observe some fraction of network traffic. In a move known as *predecessor attack* [36], a malicious entry node can attempt to enumerate many users simply by observing the connection's previous hop and comparing it to the list of all known Tor nodes obtained from the directory authority. [see 16, p. 46]

Assuming uniform node selection, the probability of an adversary either controlling or monitoring the entry node of a specific circuit is given by the fraction c/n, where c denotes the number of attacker-controlled entry nodes and n the total number of entry nodes.[4] [see 28] Thus, the resulting quotient lies in the interval from 0 to 1, where 0 denotes that the attacker controls not even a single node and 1 that the attacker controls all nodes.

Restricting the range of entry nodes to a semi-persistent set mitigates the impact of the predecessor attack, since a malicious node must be assigned the Guard flag, and then can only profile the clients that selected it is an entry Guard. [see 16, p. 28]

[2] Bandwidth measurement will be discussed in detail in Section 3.2 on page 18.

[3] This value is harder to fake, since it is not self-reported but measured remotely.

[4] However, there exists a bias in the node selection algorithm towards high bandwidth, as to be explained in Section 3.2 on the next page. Consequently, we would have to modify the calculation to use c as bandwidth of compromised nodes out of the total bandwidth of all nodes n. Nevertheless, the concept remains the same.

End-to-End Correlation Attack

Tor only minimally hides end-to-end timing correlations, and thus is known be vulnerable to end-to-end timing correlation attacks. [cf. 40, 41] In other words, if an adversary is capable of observing users' traffic entering and exiting the anonymity network (e.g. by controlling or monitoring the first and the last node of a circuit), that adversary then can deanonymize users with high probability by correlating timing and volume information, and so identify traffic patterns [42]. [see 28]

If both entry and exit nodes of a circuit are controlled or monitored by an adversary, that circuit is assumed to be compromised with regard to end-to-end correlation attacks. As previously explained in Section 3.1 on the facing page, the probability for choosing a malicious entry node is given by c/n, where c denotes the number of attacker-controlled entry nodes and n the total number of entry nodes. The same probability applies to choosing a malicious exit node. Hence, without the use of Guard nodes, the probability of choosing both malicious entry and exit nodes is given by $(c/n)^2$, which usually is quite small.

Nevertheless, since most users create many circuits over time, even this small probability accumulates that the risk of having at least one circuit compromised becomes quite large. For users for whom it is vital to avoid revealing the identity of any of their communication partners, compromising even only a fraction of their circuits will likely be enough to fatally threaten their anonymity. [see 28]

Clients that have chosen a malicious Guard node will have a c/n fraction of their circuits compromised. On the other hand, clients that have chosen a benevolent Guard will have no compromised circuits. It is generally assumed that having many circuits compromised is only marginally worse for users' security than having few. Consequently, Guard nodes improve the average security of the Tor network, since for users with good Guard nodes, the situation is much better, and for users with bad Guard nodes, the situation is not much worse than before.

Essentially, the Guard node approach recognizes that some circuits will be compromised. With regard to anonymity, it is considered more important to increase the probability of having no compromised circuits at the expense of increasing the proportion of circuits that will be compromised if any of them are, because the security difference between having no compromised circuits and having few is much greater than that between having few and having more. [see 28]

3.2 Path Selection

Tor's current path selection algorithm selects nodes according to the following listed constraints. [see 43]

- The same node may only be chosen once for the same path.
- Clients may never select two nodes from the same *family* for the same circuit. Two nodes belong to the same family, if they list each other in their family-descriptors, which may be set by the nodes' operators.

- In a path Tor clients may not use any two nodes, whose IP addresses are in the same /16 network[5]. This prevents an adversary controlling a small network to gain advantage simply by deploying many nodes. Thus, in order to avoid this basic form of Sybil attack [44], where an adversary would simply control nodes on the same machine or network, clients ensure that every node in a path has a different /16 IP address prefix.

- Each node in a path must be marked as *Running* by the directory authority, meaning that a directory authority managed to connect to it successfully within the last 45 minutes.

- The entry and exit nodes of a path must not be running a blacklisted version of Tor. Furthermore, exit nodes must not be known to behave suspiciously[6].

- The first node on a path must be marked as *Guard* by the directory authority.

Bandwidth Weighting

Initially the Tor network was composed of only a few high-bandwidth nodes and had few users. Therefore, clients assumed nodes to be alike and selected nodes of a path uniformly at random from the set of all active nodes. This method offered the maximum achievable anonymity, as an attacker could only influence the path selection of clients by operating more nodes.

However, as the Tor network grew in popularity, this approach created terrible bandwidth bottlenecks, since the Tor network became heterogeneous with regard to the nodes' available capacities - clearly not all nodes were able to push the same amount of traffic. This led to overloaded low-capacity nodes and underutilized high-capacity nodes, since nodes with weak performances were chosen with the same probability as very powerful nodes. It became necessary to ensure that the traffic load was balanced across the available bandwidth in the network. [see 28, 16, pp. 27-29]

Consequently, to better balance the traffic load across the overall available bandwidth, the current path selection algorithm weights nodes based on their bandwidth, giving a higher selection probability to nodes with more bandwidth. Thus, high-bandwidth nodes are assigned to more circuits, and therefore more of the traffic. This increases performance while still providing a reasonable level of anonymity. [see 28, p. 8]

Bandwidth Descriptors

In addition to the average bandwidth it is able or willing to relay, a node reports the peak throughput it has observed in the past day to the directory authority. [see 18] These values are called *bandwidth descriptors* [31] and are published by the nodes themselves, and so cannot be considered trustworthy. For this reason, low-resource adversaries who control only a few nodes and little bandwidth could attract traffic and increase their probability of compromising circuits by falsely reporting high bandwidth values. To mitigate the effectiveness of such an attack, all bandwidth advertisements were restricted by an upper limit[7] so that a single rogue node could

[5] According to the Classless Inter-Domain Routing (CIDR) notation.
[6] The flag *Bad Exit* is used to mark such suspiciously behaving nodes.
[7] The maximum *believable bandwidth* is currently 10 Mbit/s.

not simply claim to have infinite bandwidth. But some nodes did in fact have very high bandwidth and were unable to report it as such. Consequently, the metric based on self-advertised bandwidth descriptors was replaced. [see 28, 16, pp. 27-31]

Bandwidth Scanner

Starting with version 0.2.2.6-alpha, Tor made a paradigm change in the path selection metric. Contrary to the bandwidth descriptors metric, which is based on passive observation of bandwidth by nodes themselves, the new bandwidth scanner metric [45] depends on active measurements of nodes' available capacities. [see 22]

A subset of the directory authorities actively measure nodes' available throughput capacities, to prevent misbehaving nodes from claiming (intentionally or accidentally) to have more bandwidth than they actually do. [see 28] Based on each node's available bandwidth, the directory authority assigns selection weights and publishes those in the directory consensus. Clients then use these weights to bias node selection for their circuits in order to distribute load toward nodes with more available bandwidth.

The basic goal of actively measuring nodes' available bandwidth is to transmit data through a circuit with two nodes of similar performance according to their bandwidth descriptors. Since a node's available bandwidth is inconstant, varying over time, it is approximated by repeatedly transmitting data until every node has participated in at least five measurement circuits. [cf. 45, 22]

Bandwidth Manipulation Even though nodes' spare bandwidth is actively measured remotely, rather than relying on self-advertised values, the method is still susceptible to manipulation. Since nodes' throughput is measured over two-hop circuits solely by the directory authority, which consists of only a few centralized and well-known nodes, those measurements can easily be identified by a malicious node. Such a malicious node can then allocate more resources to these throughput measurements and thus cause its bandwidth to be overestimated, which may eventually strengthen attacks. Alternatively, it can simply drop measurement probes, so that the self-advertised information from the bandwidth descriptor is used as a fallback instead. These descriptors are reported by nodes themselves and thus can be easily manipulated, as explained previously on the preceding page. [cf. 22, 19]

Additional Restrictions

But weighting by bandwidth alone also proved suboptimal. Imagine two nodes with equal bandwidth, where the first node is suitable for use in any position of a path, but the other node only for a middle node. In this case, the first node will be chosen three times as often, causing it to be overloaded in comparison to the other node. [see 28] Therefore, as of version 0.2.2.10-alpha, clients additionally bias against selecting Guard nodes except as entry points, and exit nodes except as egress locations. [see 46]

In any case, Guard and exit nodes are considered for other positions only if the available total bandwidth of all Guard nodes or all exit nodes is at least one third of the overall available bandwidth, to ensures that sufficient bandwidth can be provided. [see 22, 16, p. 30]

3.3 Congestion Avoidance

There are several reasons for congestion in the Tor network. First, the performance of the Tor network is degraded by the bulk data transfers of file-sharing users. Furthermore, Tor's path selection strategy does not distribute load over the network optimally [see 17], resulting in the underutilization of some nodes and congestion of others.

We will describe two methods for avoiding congestion in the Tor network, Circuit Build Time (CBT) and token bucket rate limiting, both of which will later be important to understanding our approach.

Circuit Build Time

CBT is a client-side method that tries to avoid congested nodes and thus slow circuits. Some circuits are established within a fraction of a second, while other circuits take over one minute to build. These build times give a hint as to how well that circuit will perform for future traffic. [see 17]

Since version 0.2.2.8-alpha, Tor uses adaptive circuit build timeouts to drop circuits that take too long to build. In this way, extremely slow circuits are discarded early, so that clients never use them. To find a suitable timeout value, clients track the circuits' build times they observe and then recognize when a circuit has taken longer to build compared to other circuits. Thus, clients can also adapt to changing network conditions. [cf. 17, 47] To decide on a timeout value early, a priori information about the statistical distribution of build times is used. Empirically it was found that CBT values fit well to a Fréchet distribution, an extreme value distribution. Figure 3.1 on the facing page depicts the approximated Probability Density Function (PDF) of roughly one million circuit build times. Notice the long tail, which goes even further than can be shown in the Figure, up to 120 seconds. However, estimators for that distribution converge slowly and are difficult to calculate. For this reason, instead of calculating estimators for a Fréchet distribution, the tail of the Fréchet distribution is approximated with a Pareto distribution. [cf. 43] In addition to the approximated PDF of CBTs, Figure 3.1 also shows the Pareto approximation of their tail, illustrating that their 80th percentile[8] can be approximated closely with such Pareto curve.

As soon as 100 CBT values are gathered, the timeout value is calculated. But to have a better sense of expected build times over the long term, the 1000 most recent values are stored in a circular array. The timeout value itself is calculated by using the Pareto percentile function such that 80% of the mass of the distribution are below the timeout value. Thus, it is expected that Tor clients will accept the fastest 80% of all paths on the network. [see 43]

Token Bucket Rate Limiting

In order to allow node operators to specify the amount of bandwidth they are willing to provide, Tor nodes use the *token bucket* algorithm to control the rate of traffic.

Briefly, each node starts with a bucket (i.e. a fixed number) of tokens, and decrements its token count as cells are sent or received. When a node's bucket is empty (i.e the token count

[8] The 80th percentile indicates the value below which 80% of all observed values can be found.

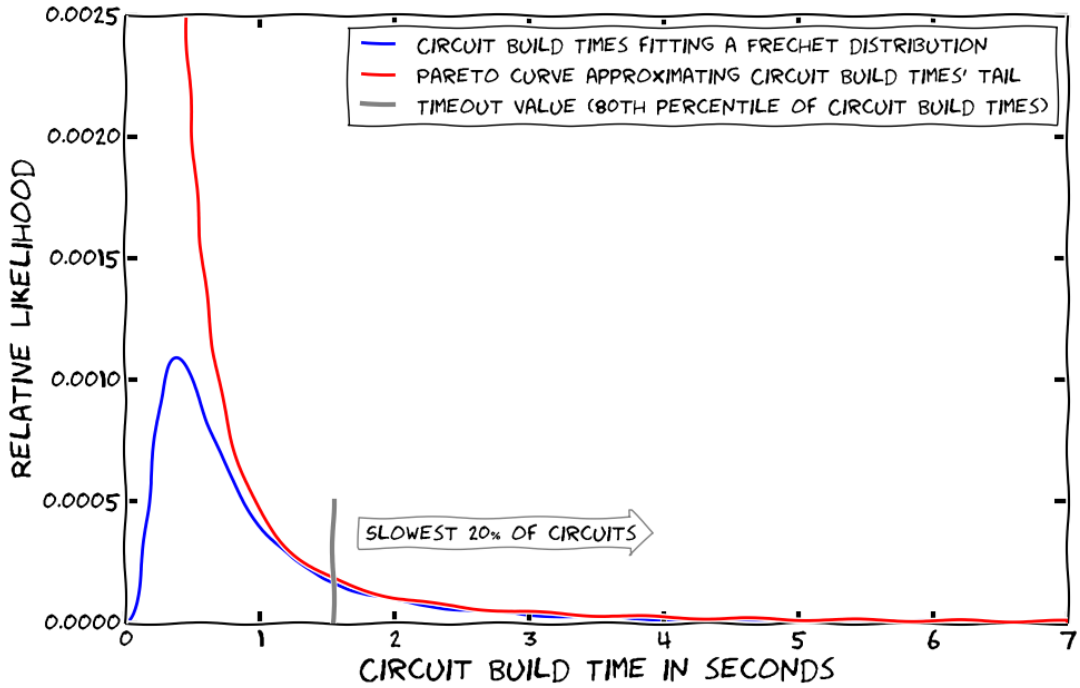

Figure 3.1: Calculation of circuit build timeout value

reaches zero), the node must wait to send or receive cells until its token bucket is refilled. [cf. 18] Tokens are refilled at a regular interval, which was reduced from 1000 to 100 ms in Tor version 0.2.3.5-alpha, in order to improve performance.

This approach enforces a long-term average rate of traffic, while still permitting short-term bursts above the allowed bandwidth. [see 28]

3.4 Tor Control Protocol

The *Tor Control Protocol* is a text-based protocol, which allows other programs to communicate with a specific Tor process, regardless of their programming languages. It facilitates extracting information from a running Tor process (e.g. to display on a Graphical User Interface (GUI)), changing Tor's configuration, and controlling a running Tor process by listening to events and sending commands. [cf. 28, 48]

Initially the functionality exposed in Tor's control protocol was simply to provide status information in a specified and machine-readable format, to make the task of monitoring and controlling a Tor process easier. Since the control protocol has proven useful to researchers experimenting with Tor, additional functionality was added. Now the control protocol can be used to control arbitrarily the path selection process, gain full control over circuit construction, and gather real-time statistics of the network status, nodes' recent bandwidth measurements,

and other characteristics. In general it is used to command and control the associated Tor client instance, e.g. during experiments. [see 28, 49]

To prevent arbitrary local processes from changing Tor's configuration, the control protocol provides authentication mechanisms so that only authorized local processes can connect. [see 28]

CHAPTER 4

Related Work

> *If we don't learn to relate, we are a failed society.*

Mary Gordon, *Love, Hate, and Everything in Between*, 2012

This Chapter, in which related work is presented, is organized as follows: In the first Section we describe papers that make use of the concept of Round-Trip-Times (RTTs). The following Section explores further work related to studying performance in the Tor network. Papers related to network adversaries are presented in the subsequent Section. Finally, we give a summary and analysis of existing approaches in the last Section of this Chapter.

4.1 RTT-Measurements

In this Section we describe three papers that make use of RTTs in various ways. All proposed techniques are based on a concept introduced by Panchenko and Renner, which will be presented first.

Link-RTT

Panchenko and Renner propose a new method to obtain the RTT of the link between any two nodes, which we will call *"link-RTT"*. They then evaluate the resulting performance when this criterion is used to select paths. Their RTT measurement method intentionally violates the exit nodes' exit policies, using *"localhost"*[1] as a dummy-destination. Since every node's exit policy

[1]The Domain Name "localhost" is reserved for special use. As defined by RFC 6761, localhost will always resolve to the respective IP loopback address, which is 127.0.0.1 in the case of IPv4. [see 50] All packets sent to

refuses connections to localhost, an error message is sent back by the exit node without contacting any further host. This ensures that only the RTT of the circuit is measured, without any additional latency. [see 21]

Tor's *"leaky-pipe"* circuit topology allows streams to exit at different nodes of a circuit, i.e. this allows traffic to exit even from the middle node. [see 28] Making use of this leaky-pipe circuit topology, the authors extend their RTT measurement technique to measure RTTs of partial circuits, aiming to infer link-wise RTTs between any two nodes. In their analysis, latency between every possible pair of nodes in the network was measured. Thus, in a network of n hosts, n^2 measurements would have to be made, limiting the scalability of this approach in practice dramatically.

The authors change Tor's path selection so that the probability of a path to be selected is increased as the sum of path's link-RTTs decreases. Using this metric, they show that latency and throughput is improved, although anonymity in this arrangement is much lower than with Tor's default path selection algorithm. Another potential disadvantage of this approach is that malicious exit nodes can identify the measurement probes and thus influence results.

Congestion-Awareness

Wang et al. introduce *"node-congestion"* as a new metric, proposing a congestion-aware path selection algorithm to reduce congestion and improve load balancing. While building circuits and using applications, clients should use a combination of two measurement schemes to evaluate the congestion of nodes locally and reject nodes that appear to be highly congested. [see 19] For active measurements, Wang et al. use a technique conceptually very similar to the RTT measurement scheme introduced by Panchenko and Renner. The basic idea is that congestion is a property of a node and its measure can be given in seconds. The authors give a procedure for clients to isolate node's congestion time from other delays, such that from propagation. Clients measure the RTT of a circuit five times immediately after circuit construction, to deduce the congestion time of each node involved. In addition to active measurements after building circuits, clients may opportunistically measure nodes' congestion, while using the circuit. These opportunistic measurements impose no additional networking overhead, as they use existing cells. The client then keeps a list of all measured nodes' congestion times. When a preemptively-built circuit is needed, clients choose the circuit with the lowest sum of congestion times. If a circuit becomes too congested during its use, as observed by opportunistic measurements, clients should stop using that circuit and instead switch to another.

Experiments on the live Tor network show that performance is improved, from the use of this node-congestion technique during path selection. Furthermore, the authors argue that the security and anonymity implications of their scheme are minimal. However, there is evidence that this path selection algorithm is too restrictive, eventually excluding most nodes. [see 52] Moreover, the fact that circuits can only be switched by interrupting and reestablishing streams limits the use of this method.

---

such address loop back inside the host, bypassing local network interface hardware, as defined by RFC 5735. [see 51] This way localhost always refers to "this computer", facilitating to access the computer's own network services without using any network interface hardware whatsoever.

Node-RTT

Aiming to increase performance and anonymity, Panchenko, Lanze, and Engel propose a new metric *"node-RTT"* for path selection. Each client measures the RTT of every node through a one-hop circuit. This technique of RTT measurement is again conceptually very similar to the scheme introduced by Panchenko and Renner. [see 22]

In terms of latency, the node-RTT method not only improves performance, but also increases anonymity compared to Tor's current bandwidth description method. However, the need for clients to perform RTT measurements for every node limits the usefulness of this approach. Furthermore, the proposed approach can be detected by nodes and thus is potentially vulnerable to attacks in which malicious nodes lie about their capacities and attract a disproportionate number of clients. Additionally, the authors aim to determine whether overloaded nodes or links are the performance bottlenecks in Tor, concluding that the nodes are the primary bottlenecks.

4.2 Performance

In this Section we present papers related to performance in the Tor network, exploring the impact of the nodes' geographic distance, delays in the network, and any adverse effects of latency reduction on anonymity. Additionally, two papers proposing replacements for the current bandwidth weight function and a new peer-to-peer bandwidth evaluation method are discussed.

Delays

Dhungel et al. perform a detailed study of delays in the Tor network. [see 18] According to the authors, the delay of a circuit can be decomposed into two parts:

- Link latencies[2]
- Queuing and processing delays within nodes.

They observe that 23% of all circuits have a delay greater than one second. Such high delay is most often the result of queuing and processing delays within nodes. Furthermore, their measurements reveal huge differences in delays introduced by particular nodes, finding that a large fraction of nodes have delays that dramatically fluctuate over time. However, such dramatic fluctuation is not often seen in nodes with very high advertised bandwidth. The authors give possible reasons for those fluctuations:

- Tor's path selection algorithm itself might play a role in the delays' variation across specific nodes.
- Other applications running on a node's machine may cause a saturation of its network link or its computational resources, such as its Central Processing Units (CPUs).

[2]Delay due to latencies between the hosts involved, i.e. client and entry node, entry and middle node, and middle and exit node of a circuit.

- Cells were often found waiting for free tokens when the node was handling a large amount of Tor traffic. As explained in Section 3.3 on page 20, the token bucket refill interval had been reduced to 100 ms. The wait for free tokens should thus have decreased noticeably since their study was conducted.

Improving Path Selection

Chen and Pasquale study the impact of geographic distance and path length on bandwidth and circuit failure rates. [see 32]

The authors calculate the geographic distance using the great circle distance formula on geographic coordinates of each node, as provided by the MaxMind's IP geolocation database [53]. They show that increased geographic distance has a slightly negative impact on throughput and reliability. Focusing on circuits with two, three, and four hops, they furthermore measured that reducing the number of hops in a circuit improves throughput only slightly. With less certainty, they also find that reliability is improved with decreasing number of hops. The authors conclude that the currently used bandwidth weighted selection strategy has the best performance overall.

Weighted Shortest Path

Akhoondi, Yu, and Madhyastha propose a new geographic-based node selection method, which they call Weighted Shortest Path (WSP). [see 54]

Using also MaxMind's IP geolocation database [53], they approximate the geographic locations of nodes and then choose paths probabilistically with a preference for shorter paths. WSP takes not only the IP addresses of nodes, but also those of the client and of the destination into account. Therefore, it cannot make use of preemptively built circuits, effectively increasing waiting time for users by several seconds. Furthermore, the WSP path selection method does not to scale well with increasing number of nodes.

Latency Attack

Since Tor intentionally imposes additional delay, the question arises if attacks based solely on RTT information exist in practice. Hopper, Vasserman, and Chan-Tin aim to determine what information is leaked by latency, and if this information has any adverse effect on anonymity. [see 55]

If a client connects to two malicious destination servers using the same circuit, then the circuit's RTT for these connections will be drawn from the same distribution, whereas other clients connecting to that servers will have different RTTs. Based on this observation, the authors develop an attack that allows two colluding web servers to link connections traversing the same circuit. Using network coordinates to compile a set of latencies from potential clients to entry nodes, an adversarial exit node is then able to reduce the anonymity set of clients based on the estimated latency. Their results suggest that Tor is vulnerable to such an attack, having classified pairs of connections with a low error rate.

Throttling Attack

Providing high performance to as many users as possible is one of Tor's primary goals. Geddes, Jansen, and Hopper attempt to ascertain whether the introduced performance improvements also lead to more successful attacks, especially ones either attempting to identify possible entry Guards or trying to reduce the set of possible clients. [see 56]

The authors introduce an induced throttling attack, where an adversary-controlled exit node is artificially throttling and unthrottling a specific circuit, leading to a recognizable pattern. The authors examine if algorithms improving throughput or responsiveness of circuits also increase the effectiveness of their latency-based throttling attack. They find that even slight improvements in the accuracy of the latency estimation result in a reduction of clients' anonymity.

Bandwidth Weighting

Self-advertised bandwidth information can be used by an attacker who runs malicious nodes to attract circuits, since nodes are chosen with a probability proportional to their bandwidth, as explained in Section 3.2 on page 17. Snader and Borisov propose a replacement $f_s(x)$ for the current bandwidth weight function such that the selection probability of a node is based on its bandwidth ranking, rather than its bandwidth itself. [see 26]

$$f_s(x) = \frac{1-2^{s \cdot x}}{1-2^s}, \text{ if } s \neq 0$$
$$f_s(x) = x, \text{ if } s = 0$$

Figure 4.1: Proposed bandwidth weight function replacement

x is chosen uniformly at random from the interval $[0, 1)$. More importantly, s is a parameter that defines the degree of bias in the node selection and can be tuned by the user. Users requiring the strongest anonymity, would want nodes to be selected uniformly at random, by setting s to 0, while those who want to bias in favor of bandwidth can use greater values of s, with the risk of having a higher number of compromised circuits. Supposing that nodes are stored in a list ordered by their bandwidth, the index of a node to select is given by $\lfloor n \cdot f_s(x) \rfloor$, where n denotes the overall number of nodes. In this case, instead of simply adding a node with very large available bandwidth and thus attracting a large fraction of circuits, an attacker must add many nodes with enough bandwidth to rank highly.

Wacek et al. compared anonymity and performance of the proposed function to the default one used in Tor. Using reasonable values for s, it performed similar to slightly worse in terms of anonymity and similar to slightly better with regard to performance, depending on the specific value of s. [cf. 46]

Although the approach by Snader and Borisov is more resilient to misreported bandwidths, it allows a passively observing adversary to distinguish users' anonymity settings. This particular partitioning attack prevented its adoption, as it would help attackers to target users who want to stay particularly anonymous. To eliminate this attack and potentially improve anonymity slightly, all users would have to use the same fixed value for s, at the same time eliminating most benefits from the proposed function.

Bandwidth Measurement

As explained in Section 3.2 on page 17, low-resource adversaries controlling only a few nodes and little bandwidth can attract traffic and increase their probability of compromising circuits by falsely reporting high bandwidth values. For this reason, a method resistant to manipulation by malicious nodes is needed, to evaluate nodes' bandwidths. To overcome the currently used self-advertised and centrally measured bandwidth scheme [45], which puts additional load on the already highly loaded directory authorities, Snader and Borisov propose a peer-to-peer bandwidth evaluation called *"Eigenspeed"*, where all nodes exchange their observations of nodes' bandwidths, building a common consensus. [see 57]

The proposed bandwidth measurement scheme adds little overhead to the network, as it uses opportunistic observations based on regular interactions between nodes, making it harder for malicious nodes to manipulate. Building a common consensus evaluation is helpful to ensure that all nodes have the same view on each node's available bandwidth. It further helps all clients to maintain similar behavior, preventing partitioning attacks, where a single client's view of the network can be manipulated for further exploitation. The observed throughput values are not meant to represent the potential bandwidth capacities of nodes. It rather provides a starting point for their evaluation, but can be too sparse, if a node communicates with only a subset of other nodes. To address this problem, not only the node's own observations but also those of other nodes are taken into account. The consensus bandwidth estimation is achieved by combining the observations across multiple nodes using Principal Component Analysis (PCA).

Unfortunately this method has some problems. As low-bandwidth nodes are limited by their own bottleneck link, they have difficulty monitoring high-bandwidth nodes. Furthermore, the authors' bandwidth estimation method does not support asymmetric upload and download bandwidths. In the end, the authors find their method less accurate than using self-advertised values from the descriptors.

4.3 Network Adversaries

Like any other low-latency anonymity network, Tor is known be vulnerable to end-to-end timing correlation attacks, as explained earlier in Section 3.1 on page 17. In contrast to relay adversaries, network adversaries do not run nodes in the hope that clients will choose their malicious nodes at the entry and exit positions of circuits. Instead, network adversaries leverage their position as carrier of network traffic to correlate Tor traffic that crosses their network at critical points between client and entry node, and exit node and destination. Research on network adversaries has focused on country-, Autonomous System (AS)-, and Internet eXchange Point (IXP)-levels.

Country-Level Adversary In the face of country-level adversaries, it is generally held that anonymity is improved by ensuring that organizations such as governments are not able to observe both the entry and exit segments. Thus, to ensure that such a country-level adversary is not capable of simultaneously monitoring enough points in the network to break users' anonymity, paths should be selected to go through as many countries as possible, likely limiting traffic monitoring capabilities to particular jurisdictions. [cf. 58, 32]

AS-Level Adversary Network connections between host computers on the Internet are very rarely direct connections. Rather, the Internet is composed of tens of thousands independent networks, so-called ASes. As data is relayed from one host computer to another, it traverses a sequence of ASes. Since the forward and reverse paths between two hosts on the Internet are often asymmetric, data possibly traverses a different sequence of ASes in the forward and reverse directions. If a common AS appears across both ends of a connection, i.e. from the client to the entry node and from the exit node to the destination, then an observer located at that AS can perform statistical correlation attacks, potentially identifying client and destination. Hence, an adversary controlling or observing traffic entering or leaving the AS can deanonymize clients whose communications originate and terminate within that AS, regardless of where Tor nodes are placed. [cf. 59, 39, 54]

IXP-Level Adversary Providing connectivity to present Internet Service Providers (ISPs), IXPs allow exchange of traffic between peering ASes, usually at a cost savings or performance improvement compared to sending traffic via a transit provider. Therefore, an IXP is in a position to see all traffic flowing between its peering ASes. While it appears at the AS-level that a path makes multiple transitions between distinct ASes, it might still pass through a single IXP. Therefore, even if a path attains high AS diversity, a single entity might remain who is able to deanonymize traffic. [cf. 39, 58]

Location Diversity

Bauer observes that the vast majority of Tor traffic is handled by a small set of nodes centralized within only a few countries. Germany and the United States together contribute nearly 59% of all running nodes. In terms of overall bandwidth, nodes located in those countries represent as much as 68%. This has significant implications to Tor's anonymity properties, as location diversity reduces the ability of a single country-level adversary to conduct traffic analysis. The author concludes that, given the current bandwidth distribution, location diversity is currently impossible to guarantee while maintaining adequate traffic load balancing. [see 16, pp. 62-64, 78-79]

AS-Diversity

Feamster and Dingledine investigate the AS-level adversary's ability to infer which parties are communicating. By analyzing Border Gateway Protocol (BGP) data, they roughly estimate 10 to 30% of all paths to go through single ASes, showing that country diversity is not sufficient to maintain anonymity in the face of an adversary with the ability to control an ISP. [see 60]

Although their investigation was conducted at a time when the Tor network consisted of only very few nodes and they only considered clients located at a handful of consumer ISPs within the United States and a small set of arbitrarily chosen websites, the paper brought valuable insights. While aiming to evade country-level adversaries, they argue that selecting paths that go through many countries may adversely affect anonymity, since paths that traverse many ASes are more likely to have the same AS on both sides of the path. They suggest maximizing AS

diversity when selecting paths, and thus reduce the likelihood that any single ISP can observe connections. [cf. 59, 39, 58]

AS-Awareness

Edman and Syverson also examine AS path diversity in the Tor network, analyzing the potential threat of AS-level adversaries. [see 59]

Based on traffic data collected from public Tor nodes, they examine the distribution of client origin and destination ASes, concluding that for both source and destination, less than two percent of the ASes accounted for over 50% of the connections. More importantly, they show that in 2008 the ASes of clients and destinations were significantly different from those supposed by Feamster and Dingledine in 2004. Consisting of a more accurate algorithm for inferring AS-level routing paths and a larger set of BGP routing data, they propose a simple AS-aware path selection algorithm to avoid AS-level traffic correlation attacks. Despite avoiding full AS-level path inference, their algorithm is very expensive in terms of both computational complexity and storage requirements.

Overall, they find a single AS able to observe almost 40% of all forward or reverse paths, which is significantly greater than the 10 to 30% previously suggested by Feamster and Dingledine. Furthermore, the authors compared the effectiveness of different approaches to reduce the average chance of AS-level path compromise. They find country separation to be the easiest and lowest overhead approach and Tor's distinct /16 subnets policy to be largely effective, but determine that their path approximation method is the most effective approach. In any case, they show that no matter how Tor routing is done, AS-level adversaries are a largely unavoidable threat to Tor. Furthermore, the authors suggest that clients that typically do not change AS much, should use Guard nodes within their local ASes. Those clients would then be immune to AS-level attacks except by their home ASes.

LASTor

Akhoondi, Yu, and Madhyastha propose a more sophisticated heuristic for predicting paths on which an AS can correlate traffic between the entry and exit segments. [see 54]

Since it is difficult to infer exactly AS-level routes between pairs of arbitrary IP addresses, the authors instead aim to predict the set of ASes through which the traffic is routed. They develop an alternative Tor client, which they call *LASTor*. It aims to predict the set of ASes through which the traffic will flow and then avoids paths on which an AS can correlate traffic between the entry and exit segments. While measuring a false-negative ratio of 57% with the default Tor client, they find their LASTor client failing to identify only 11% of the paths where a common AS exists at both the entry and the exit segments.

LASTor takes not only the nodes' IP addresses into account, but also those of the client and the destinations. Hence, in accordance with their WSP algorithm described on page 26, LASTor can not make use of preemptively built circuits, effectively increasing waiting time for users by several seconds. This impedes the authors' efforts to reduce the algorithm's runtime. Furthermore, we do not find a description of the algorithm's behavior when client usage involves several

destinations IP addresses, common when browsing websites, for example. Using a different circuit for each destination would reduce client anonymity. The same is true for using the same circuit when destinations are geographically far apart.

IXP Attack

Murdoch and Zieliński show that IXPs can be used to monitor packets in Tor flows even when AS diversity is assumed. [see 58]

The authors argue that AS diversity is an insufficient defense against traffic correlation attacks by IXP adversaries, since traffic is routed between ASes at IXPs. Hence, a single IXP may observe traffic traversing multiple ASes and still could not be detected at the BGP level. AS-diverse paths will not substantially impact crossings on big IXPs. For example, they find that the London INternet Exchange (LINX), one of the largest IXPs in Europe in terms of throughput, is present on 27% of all paths, despite being invisible at the BGP level. Hence, an adversary exploiting an IXP as an observation point would be capable of monitoring a substantial quantity of traffic through the Tor network.

High-end networking devices located at IXPs are already equipped with traffic monitoring features, which are commonly used in practice to aid network management. Protocols such as NetFlow[3] are used to record aggregated data on traffic passing through network equipment. These protocols could also be used by an adversary to perform traffic analysis and so effectively trace users, since a significant number of flows pass through an IXP. In their paper, the authors only inspect incoming and outgoing traffic and do not make any assumptions about the structure of the anonymity network. Assuming that the attacked flow passes through an attacker controlled IXP on its path both into and out of the anonymity network, each flow is sampled and the attacker computes the probability that the input flow corresponds to the output flow. The authors use a Bayesian approach to show that an IXP-level adversary could sample traffic from multiple ASes and correlate flows, effectively deanonymizing users.

They conclude that more flows provide more protection, that the effectiveness of their attack depends only on the number of sampled packets, and that even random delays of reasonable length do not protect against their attack.

IXP- and AS-Adversaries

Previous work has considered network adversaries that control either a single AS or an IXP, implicitly assuming that malicious ASes or IXPs are non-colluding. But organizations, such as corporations, intelligence agencies, or countries, might be in a position to observe network traffic or control several geographically diverse ASes or IXPs. Analyzing the threat of IXPs and IXP coalitions, Johnson et al. explore Tor's vulnerability to traffic correlation attacks from network adversaries that control one or more fixed ASes or IXPs. [see 39]

Although a network adversary could also delay, alter, drop, or add communication in a variety of ways, the authors limit their study to the adversary's ability of passive observation.

[3] NetFlow equipped devices record so called "flows", unidirectional sequences of packets with some common properties that pass through such a networking device. A flow record includes details such as source and destination IP addresses, source and destination ports, packet and byte counts, timestamps, Type of Service (ToS), etc. [see 61]

Simulating different user behavior and assigning users to the five most popular client ASes, the authors evaluate how long a user can stay anonymous, and present the probabilities of compromise over time. To quantify anonymity in the face of traffic correlation attacks, the authors explore path compromise rates and the time to compromise using several user models.

In the face of an AS adversary controlling a single AS, results show that almost any user type uses at least one compromised circuit within a few months. If that adversary controls a second AS, the time to first compromised circuit is drastically reduced, showing the dramatic effect an adversary that controls multiple ASes can have on users' anonymity. IXP- and AS-adversaries have similar results in the worst case. But IXPs are significant less of a threat in the best case, because 80% of the network links do not traverse IXPs. Nevertheless, the complexity of performing traffic correlation at an IXP is likely to be significantly lower than at an AS, since ASes may span large regions while IXPs are in individual locations. Even though IXPs are less of a threat, it is easier for them to perform traffic analysis and therefore should be considered as adversaries; especially organizations that control multiple IXPs.

Simulation results show that users' anonymity varies significantly with their location and especially with their selection of Guard nodes. Since the path between client and its entry Guard is relatively fixed, either the adversary only needs to wait until it is also able to observe the exit side, or no compromise of the client's streams will be possible until the client selects new Guards.

4.4 Summary and Analysis of Existing Approaches

Performance

Chen and Pasquale showed that reducing both geographic distance and path length improve performance only slightly, but may have yet unknown security implications. Therefore, both methods are unsuitable for achieving generally applicable performance improvements.

According to Dhungel et al. circuit delays in the Tor network can be decomposed into link latencies between nodes and queuing and processing delays within nodes. They find that the queuing and processing delays within many nodes fluctuate dramatically over time. Therefore, we will focus on a method that allows us to avoid overloaded nodes when we want to use them.

In general, measurements can either be done by clients or by centralized network monitors. The first possibility adds more additional load to the network, but clients do not need to trust any other parties. The latter suffers from the fact that there is a single point of failure. [cf. 21] Since delays within nodes fluctuate dramatically over time, a centralized model becomes unfeasible, because it would not be possible to react fast enough to changing conditions. Furthermore, it would also mean increasing consensus data. Therefore, we aim for a client-centric, decentralized model.

Geddes, Jansen, and Hopper find that algorithms improving throughput or responsiveness of circuits also increase the effectiveness of latency-based attacks. Even slight improvements in the accuracy of the latency estimation result in a reduction of clients' anonymity.

RTT-Measurements

In the beginning of this Chapter we presented three different papers that make use of RTT-measurements. Those approaches aim to improve Tor's performance by reducing latency. The techniques used are based on a concept introduced by Panchenko and Renner.

According to Wacek et al. active RTT-measurement is the most effective proposed improvements to path selection. [see 46] We will also use Panchenko's and Renner's method to measure RTTs, utilizing a violation of exit nodes' exit policies caused by using *"localhost"* as a dummy-destination. This ensures that the RTT measured is indeed that of the circuit, without extra latency. A potential disadvantage of this approach, however, is that malicious exit nodes could identify the measurement probes and influence the results.

In contrast to the related works, we do not make any assumptions about whether bandwidth and latency should be treated as property of a node or of a link.

Network Adversaries

Tor currently does not implement any method that has been explicitly designed to protect against network adversaries operating on country-, AS-, or IXP-levels. The only method that provides some kind of protection against such network adversaries is Tor's /16 subnet restriction when selecting paths, as explained in Section 3.2 on page 17. This constraint does not seem to be effective enough, however, since between 60 to 70% of all ASes have at least two nodes with IP addresses in different /16 subnets. [see 59, 54] According to Akhoondi, Yu, and Madhyastha, the default Tor client fails to identify 57% of the paths, where a common AS exists at the entry and exit segments. [see 54] Moreover, some paths are erroneously avoided, because nodes have IP addresses in the same /16 subnet, although those nodes are in distinct ASes.

Given the current bandwidth distribution, country diversity is currently impossible to guarantee, while maintaining adequate load balancing of traffic, as Bauer noted. [see 16]

By analyzing BGP data, Feamster and Dingledine roughly estimate 10 to 30% of all paths to go through single ASes, indicating that country diversity is not sufficient to maintain anonymity in the face of an adversary with the ability to control an ISP. [see 60] Edman and Syverson find a single AS able to observe almost 40% of all forward or reverse paths. [see 59] More recently, a study by Wacek et al. indicates that the same AS may appear on both sides of about 28% of all paths. [see 46] Comparing different approaches to reduce the average chance of AS-level path compromise, Edman and Syverson find country separation to be the easiest and lowest overhead approach, Tor's distinct /16 subnets policy to be largely effective, but their path approximation approach to be the most effective approach. Nevertheless, they show that no matter how Tor routing is done, AS-level adversaries are a largely unavoidable threat to Tor. [see 59]

Murdoch and Zieliński show that an adversary exploiting an IXP as an observation point would be capable of monitoring a substantial quantity of the Tor network's traffic. They conclude that even random delays of reasonable length do not protect against their attack; only more flows provide more protection. [see 58]

Johnson et al. show the dramatic effect that an adversary controlling multiple ASes or IXPs, such as a corporation, intelligence agency, or country, can have on users' anonymity. Furthermore, they argue that security metrics should be defined in terms of such adversaries and should present the probabilities of compromise over time. [see 39]

Conclusion In addition to the proposed solutions that we already mentioned, Juen proposes a new selection algorithm that aims to provide both AS- and IXP-diversity. [see 62] However, using such path selection algorithms may be problematic, as protecting against one type of network adversary potentially can have adverse effect on defenses against other threats. Moreover, as an official path selection restraint, it becomes more difficult when the country or AS of both the client and the final destination should be considered.

Because of those yet unresolved issues, we will not consider network adversaries in the anonymity metric in this thesis, but instead use the classic entropy model, as will be explained in Section 5.5 on page 46. However, future work should consider security metrics in the terms of such network adversaries and should present probabilities of compromise over time.

Guard Node Protection Users' anonymity varies significantly with their location and especially with their selection of Guard nodes. Since the path between client and its entry Guard is relatively fixed, an adversary observing the user's connection to the Guard, such as an ISP, needs to wait until it is also able to observe the exit side. If that adversary does not exist between the client and its Guards, it will not compromise any of the client's streams until the client selects new Guards. Increasing the Guard node expiration time would significantly increase the time to compromise. Reducing the number of Guard nodes would also have a positive impact on anonymity, as suggested by Elahi et al. and confirmed by Johnson et al. [see 37, 38, 39].

Therefore, we find it important to consider Guard nodes, as they have great influence and are known to provide valuable protection, as explained in Section 3.1 on page 15.

CHAPTER 5

Methodical Approach

> *Don'ts of Mathematical Modeling*
>
> - *Don't believe that the model is the reality*
> - *Don't extrapolate beyond the region of fit*
> - *Don't distort reality to fit the model*
> - *Don't retain a discredited model*
> - *Don't fall in love with your model*
>
> S. W. Golomb, *Mathematical models - uses and limitations*, 1970

Since Tor intentionally bounces traffic around the world several times, the best achievable latency is by design higher than that achieved on regular Internet connections. Although both end-to-end latency and throughput in the Tor network have been improved significantly over the last few years [see 16, pp. 67-70], users still often experience long delays [cf. 17, 18, 19].

There are several reasons that latencies are unacceptably high for delay-sensitive, interactive web users. First, the bulk data transfers of file-sharing users consume a disproportionately large amount of the network's bandwidth, congesting particular nodes. Nevertheless, only about half of the overall bandwidth available in the Tor network is actually used. This is on the one hand more than in most reasonable networks, but on the other hand shows that there is significant room for alternative methods that better utilize the available resources to improve performance.[1] [cf. 16, 22, p. 78]

[1] Tor's current selection algorithm is optimal when the network is fully loaded. If the network is not fully loaded, fast nodes end up with relatively less load than slow nodes. [see 17]

As shown in Chapter 4, various techniques were proposed in recent years to achieve lower latency and higher throughput while maintaining anonymity. Wacek et al. evaluated some approaches and concluded that a combination of several techniques would provide the best results. [cf. 46] We will focus in particular on one technique, which we call *Circuit-RTT*. We use a common definition of RTT as the sum of two times the latency plus the processing delay, where latency is the time between the first bit leaving the sender and the last bit arriving at the receiver.[2]

5.1 Used Concepts

In our attempt to reduce latency in the Tor network, we actively measure the time it takes for data packets to go through a Tor circuit and back by violating the exit policy of the exit node. This active measurement approach is based on a technique originally proposed by Panchenko and Renner [21] in 2009, as presented in Chapter 4 on page 23, later also employed by Wang et al. [19], and Panchenko, Lanze, and Engel [22] in 2012.

Currently, only information that is known to all network participants, such as the estimated spare bandwidth of nodes, is used to influence path selection, as explained earlier in Section 3.2 on page 17. The only exception to this is the CBT method, described in Section 3.3 on page 20, which is used to decide whether a circuit should be used after it has been established.

Circuit Build Time (CBT)

Some circuits are established within a fraction of a second, while others take over a minute to build. Since version 0.2.2.8-alpha, Tor clients drop circuits that take too long to build. By calculating and continuously adapting a timeout value, circuits that have build times within the slowest 20% are discarded even before a client can even use them. Hence, it is expected that clients accept the fastest 80% of circuits, in terms of latency.

Figure 5.1 on the facing page shows the timing associated with building a circuit. First, to establish an encrypted tunnel with the entry node, the client negotiates a shared symmetric session key. Subsequently, to extend the encrypted tunnel further to the middle node, the client connects to it through the encrypted tunnel previously established and negotiates another shared symmetric session key. Finally, through the newly established tunnel, another shared session key is negotiated with the last node, completing the bidirectional, real-time virtual circuit of layered encryption.

While establishing a circuit, data is sent through the involved nodes several times. Hence, a circuit's build time consists of multiple times the latencies between these nodes. It additionally includes the processing delay, the time spent by the nodes both on forwarding packets and on computing the encryption keys. The latter especially can be on the order of seconds if a node is computationally overloaded.

[2] In more detail, latency is the sum of transmission time and propagation delay. Transmission time denotes the time between the sending of the first and the last bit. Propagation delay denotes the time it takes for one bit to travel from the sender to the receiver.

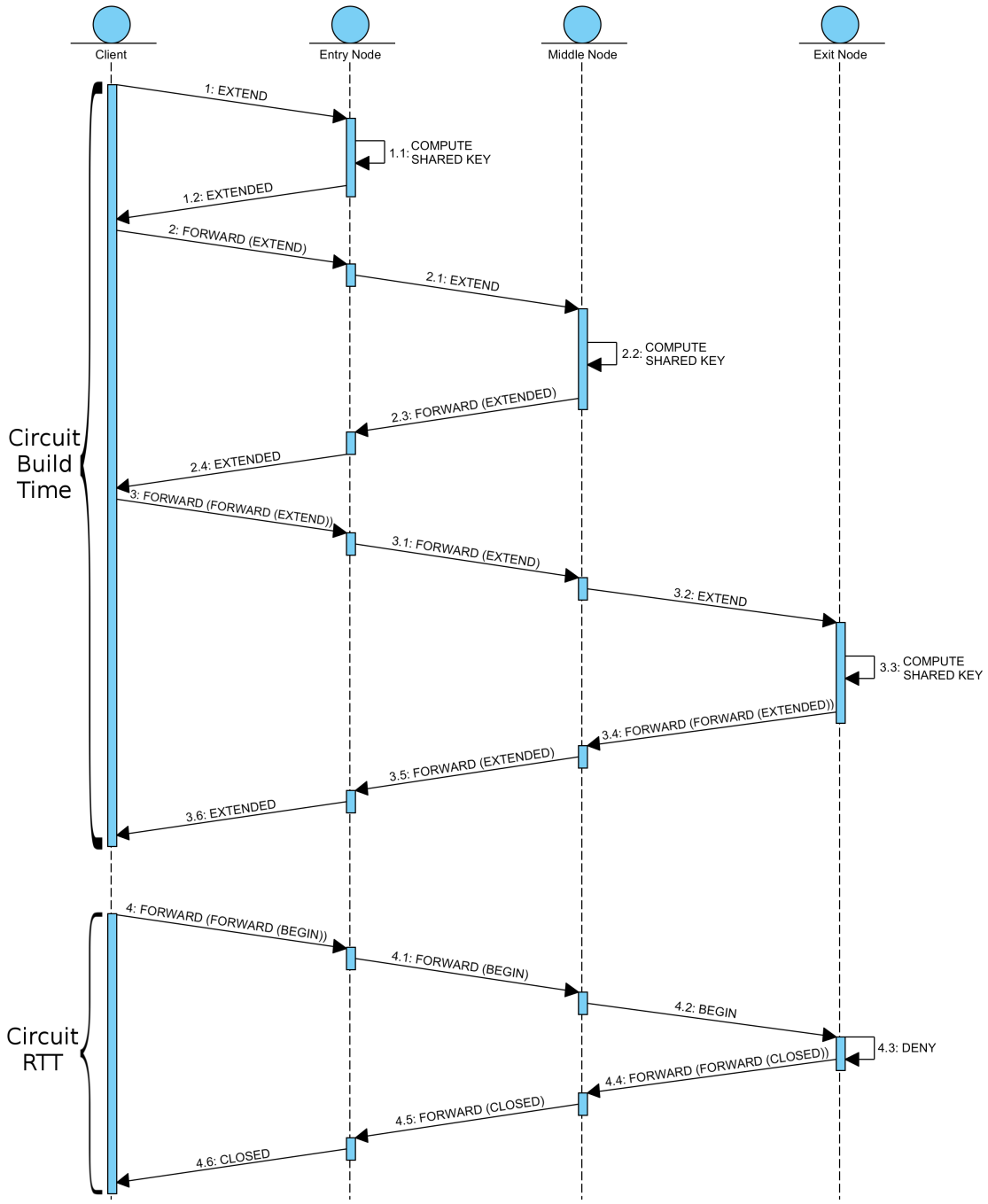

Figure 5.1: Timing associated with CBT and Circuit-RTT

Circuit-RTT

After a circuit has been successfully established, the Circuit-RTT method aims to measure only the RTT of that circuit by asking the exit node to open a TCP connection to an IP address from the 127.0.0.0/8 network. Since all nodes refuse such connections due to their exit policies, the request results in an error, and the exit node sends a corresponding error message back without contacting any further host. Our timing client measures the interval between sending the request and receiving the reply. The processing delay, the time needed for forwarding packets, is negligibly small, so the measured time interval can be considered very close to the real RTT of the circuit.

Tor Control Protocol

We use the Tor Control Protocol, described in Section 3.4 on page 21, to control the Tor client's behavior during experiments. Since the client is operated by us, we are able to manage the client's path selection and circuit building process, and control which TCP streams are associated with certain circuits.

Aim of the Work

In this thesis we aim to find an improvement for latency, the most important metric from the users' perspective [see 63]. To validate our assumption that Circuit-RTT is a good metric for latency, we examine the correlation both of RTT and Time-To-First-Byte (TTFB) and of CBT and TTFB. Furthermore, we examine how the RTTs of circuits change over time and whether the client's selection of Guard nodes changes the observed RTTs significantly. More importantly, we measure the influence pruning the slowest circuits according to the CBT and Circuit-RTT methods has on latency, bandwidth, and also on anonymity, since a latency optimized strategy can only be considered acceptable if anonymity is not decreased significantly. Our measurement methods are described in detail in Section 5.5 on page 44.

5.2 Path Generator

Since building a circuit takes anywhere from several tenths of a second up to two minutes, establishing and measuring millions of circuits one after the other would take months. Instead of sequentially building circuits, we first select paths and then build and measure circuits concurrently. This reduces the time required for running our experiments tremendously. Since the selection of paths is only a small burden on local processing power, but building and measuring a circuit involves network Input/Output (I/O), which takes the most time overall, we decrease the time needed to run our experiments by generating paths sequentially and building circuits in parallel. Path generation will be discussed in detail in this Section and our parallelization approach will be presented in Section 5.3 on page 40.

Existing Path Simulators

At the time of writing, three software projects exist, that could be used as Tor path simulators:

TorCtl TorCtl [64] is a Python library that implements the Tor Control Protocol with extensions to support path building. These extensions include various restrictions on complete paths, such as one avoiding ocean crossings, as well as on single nodes, for instance to avoid nodes from particular countries. Since it reimplements Tor's algorithms, it is difficult to assure logical equivalence. Furthermore, it has been deprecated since 2012.

Changing of the Guards (COGS) For their paper on Guard node security [38], Elahi et al. created a simulation framework they called COGS, which is designed to simulate path selection over long time periods, and for this purpose makes use of saved Tor consensus data. In their experiment the authors examine how different algorithms for selecting entry Guards affect anonymity and network throughput. Tor's source code is modified so that it only creates paths and does not build circuits, among other changes. Unfortunately, these changes are so intrusive overall that it is difficult to verify logical equivalence to the original algorithm.

TorPS Johnson et al. built a path selection simulator, called *TorPS* [65], to predict path selection when certain parameters are changed. Like COGS, it was also designed to simulate path selection over long time periods and so makes use of saved Tor consensus data, as well. Since it reimplements Tor's path selection algorithm, its logical equivalence is also hard to verify.

Path Generator

Since we want to test circuits on the live network, we have no need to use historical consensus data. Our research has different requirements for path generation than previous path simulation tools could accommodate. Above all, we require a path generator that works identically to Tor's default path selection, described in detail in Section 3.2 on page 17, including its bandwidth weighting and constraints such as /16 subnet restriction.

We do therefore not reimplement Tor's path selection algorithm, but instead make small changes to Tor's source code that allow our Tor client to generate paths without actually building circuits. This enables us to maintain high confidence in the logical equivalence to Tor's default path selection algorithm and also makes our changes portable to other Tor versions with little effort. To facilitate the separation of path selection from circuit building, we implement two new controller commands, *"DUMPGUARDS"* and *"FINDPATH"*, which are the core components of our path generator.

DUMPGUARDS On the one hand, we require behavior identical to Tor's default path selection algorithm, which involves Guard nodes. On the other hand, we require different entry nodes for each path, because we want to probe more than a single entry node. Unfortunately, the node selection probability for middle and exit nodes depends on whether Guard nodes are enabled or not. For this reason, simply disabling the use of Guard nodes is not an option in our case. Since previously the only way to make a Tor client select a new set of Guard nodes was to delete

the Tor client's state file and restart it then, we implement a new controller command *"DUMPGUARDS"* such that the client's selection of Guard nodes can be explicitly altered. When the client receives the *"DUMPGUARDS"* command, it clears its list of currently used Guard nodes, without requiring a restart.[3]

FINDPATH Since there was no previous way to make Tor select a path without actually building a circuit from it, we implemented a new control command *"FINDPATH"* that tells the Tor software just to select a path, print it, and remove it afterwards.

Correctness We verified the correctness of our *"DUMPGUARDS"* command by examining Tor's corresponding debug messages[4], and additionally observed changing entry nodes for subsequent paths. To ensure high confidence in the correctness of the path generator using our *"FINDPATH"* command, we made only small, non-intrusive changes to Tor's source code. Furthermore, we compared the distribution of nodes in entry, middle, and exit positions on paths to the nodes' consensus weights and verified that all nodes had the correct flags assigned.

5.3 Design Methods

All measurement scripts are written in the Python programming language, which allows us easily to interface with our Tor client through its control port using "Stem" [66], a Python controller library that implements Tor's directory and control specifications.

Figure 5.2 on the next page sketches the basic algorithm used to gather the required data for further analysis. The main thread handles path selection and coordinates all worker threads. It waits until it knows enough server descriptors and then ensures that the client's Guard list is empty by executing our *DUMPGUARDS* control command. Subsequently, using our *FINDPATH* control command, the associated Tor client chooses a path and returns it. The main thread can immediately begin finding another path to probe after it has spawned a worker thread, because worker threads run asynchronously from the main thread. Each of these concurrently running worker threads receives only one path from which to build a circuit and then measures the build time, the RTT, the TTFB, and the bandwidth of that circuit. As soon as a worker thread finishes its measurements, it returns the data to the main thread. The main thread and all worker threads can simultaneously use the same control connection to the associated Tor client.

Parallelization

From a client's point of view, most of the time building and testing circuits is spent waiting for network I/O. To reduce the time required for our experiments, we parallelize all parts that involve network I/O in our measurement software. This approach uses up far less resources in terms of memory consumption compared to using multiple Tor clients. Nevertheless, small parts of the code are mutual exclusively executed by a single thread:

[3]This patch was merged into Tor version 0.2.5.1-alpha, but the name of the command changed to *"DROPGUARDS"*.

[4]For performance reasons, this was disabled during live experiments.

Algorithm 5.3.1: CIRCUIT PROBING()

```
procedure WORKER(path)
  CONNECT TO TOR CLIENT()
  BUILD CIRCUIT(path)
  RTT PROBE: CONNECT TO 127.0.0.0/8()
  TTFB PROBE: FETCH HEADERS FROM HTTP://GOOGLE.COM()
  BANDWIDTH PROBE: FETCH HTTP://TORRTT.INFO WEBPAGE()
  return (CBT, RTTs, TTFBs, Bandwidth)


main
  CONFIGURE TOR CLIENT()
  while NOT FINISHED()
      do { WAIT FOR DESCRIPTORS()
           DUMPGUARDS()
           FINDPATH()
           RUN WORKER THREAD (ASYNCHRONOUSLY)()
```

Figure 5.2: Circuit Probing

- Path generation in the main thread is run sequentially, to ensure that the previously described *"DUMPGUARDS"* and *"FINDPATH"* controller commands are executed in the right order without being interrupted. To ensure that individual measurement results are not mixed, only a single worker thread is given write access to the output file. However, the time needed for both generating a path and writing a single measurement result to disk is negligibly small.

- Circuit identifiers are used to associate circuit event messages, to particular circuits. Such an identifier can not be specified through the Tor control protocol; only the Tor process itself may specify it. We can not use the path as an identifier either, since Tor's *"LAUNCHED"* circuit event messages do not hold a path element by which we could identify the circuit. For this reason, the launching of a circuit must be exclusive to a single worker thread, so that the received circuit identifier can then be used to correlate corresponding circuit event messages. However, the time needed for launching a circuit is negligibly small.

- Although we cannot know the Tor client's internal stream identifier for our RTT measurement's TCP stream in advance, we can completely run our RTT measurements in parallel, as we use unique destination IP addresses from the 127.0.0.0/8 network as identifiers.

We can do that neither for the TTFB- nor for the bandwidth-probes, however, since we connect to servers on the Internet whose destination IP addresses we cannot choose. Thus, the start of TTFB and bandwidth measurements must be exclusive to a single worker thread, in order to gather the stream identifier chosen by the Tor client. The time needed to start a TCP connection and send it to the client through SOCKS is very short in comparison to network latency.

Hammering Protection

Since the Tor network can be extremely fragile when stressed with too many circuit creation attempts at once, and our method scales from several hundreds to thousands of concurrent worker threads, we have to avoid interfering with the live Tor network functionality. In order to affect the Tor network as little as possible, we keep a list of nodes that are probed at any given point in time, and queue new paths if they include nodes from that list. In this way, each node is probed no more than once at any point in time. We set the size of the waiting queue to two times the maximum number of concurrent worker threads. If this queue is full, no further worker threads are spawned until another worker thread finishes and a path from the waiting queue can be used. This limits the time between choosing a path and probing the circuit, ensuring that the information on the nodes involved is still valid.

Output Format

For each measured circuit, detailed information on the nodes involved, such as the nodes' server descriptors and network statuses, is stored. Furthermore, all circuit event messages, some stream event messages[5], the circuit's build time, the TTFB, and the bandwidth measurements are stored. The data is first serialized and then compressed using the Lempel-Ziv-Oberhumer (LZO) algorithm. LZO offers a very good compromise between the achievable compression ratio and the computational resources required. This ensures that our measurement results are not influenced by excessive use of computational resources for compressing data on the measuring host. The compressed data is then appended to a tar (tape archive) file. To avoid any potential filesystem limitations and to make the task of post-processing easier, the tar file is closed and a new one is opened as soon as its size exceeds 1 GB.

5.4 Experimental Environment

We used the source code of Tor version 0.2.3.25, which was the latest stable version when we started our experiments, applied our patches, and compiled a statically linked binary executable. The patches applied include both the previously described *"DUMPGUARDS"* and *"FINDPATH"* commands for our path generator and two additional, smaller patches:

[5]The circuit's RTT is calculated during post-processing from these stream event messages. Furthermore, information on circuit build failures can be gathered from circuit event messages.

Descriptors Patch By default, Tor does not start building circuits before it knows at least 75% of all exit nodes and at least 75% of nodes altogether. Since our measurements should run on potentially as many paths as possible, we raise both limits to 95%.[6]

CBT Patch Originally, we wanted to approximate the circuits' build times by measuring the time difference in the corresponding circuit event messages, i.e. the time between the first *"EXTEND"* and the last *"EXTENDED"* event. When comparing the CBT values gathered using this method to the values provided by Tor's debug messages, we found a deviation of about 1%. Although this difference could be considered negligible, we decided to write a minor patch that allows us to extract the CBT information from a Tor client using its general log messages.

Tor Settings In deviation from Tor's default settings, we changed the following configuration variables:

- *"LearnCircuitBuildTimeout"* was unset, to disable CBT learning. Hence, instead of having Tor tear down circuits when they take longer than the adaptively computed timeout value, we use a fixed timeout value of 120 seconds.
- *"MaxCircuitDirtiness"* defines the maximal time span a circuit can be used for new TCP streams after its first use. Measuring RTTs and TTFBs multiple times over a circuit may take longer than the default setting of 10 minutes. Hence, we increase the value of *MaxCircuitDirtiness* so that the circuit is not torn down prematurely.
- *"__DisablePredictedCircuits"* and *"__LeaveStreamsUnattached"* were set to make the Tor client neither build any circuits for user streams nor attach any streams to circuits itself.
- We set *"SocksListenAddress"* to 127.0.0.1, so we can use SOCKS to tunnel our RTT, TTFB, and bandwidth probing streams through Tor.
- To have access to the nodes' full server descriptors, which is necessary to evaluating a node's exit policy, we unset *"UseMicrodescriptors"*.
- Additionally, the *"MaxClientCircuitsPending"* configuration variable could be important. It defines the maximum number of circuits that have begun to be, but have not yet been completely constructed. By default, a maximum of 32 circuits may be in such a state. However, when using more than 32 worker threads, this value should be raised - up to the current highest accepted value of 1024.

PlanetLab

In order to carry out our measurements, we deploy Tor clients on several hosts from PlanetLab [27], a testbed for computer networking research consisting of over 1000 nodes distributed around the world. Figure 5.3 depicts the geographical distribution of PlanetLab sites. Using distributed hosts allows us to examine the influence of the client's location on Circuit-RTTs.

[6] Originally, we wanted to wait until the client knows all descriptors, but we encountered an issue in the Tor code that may hang a client when fetching the final descriptors.

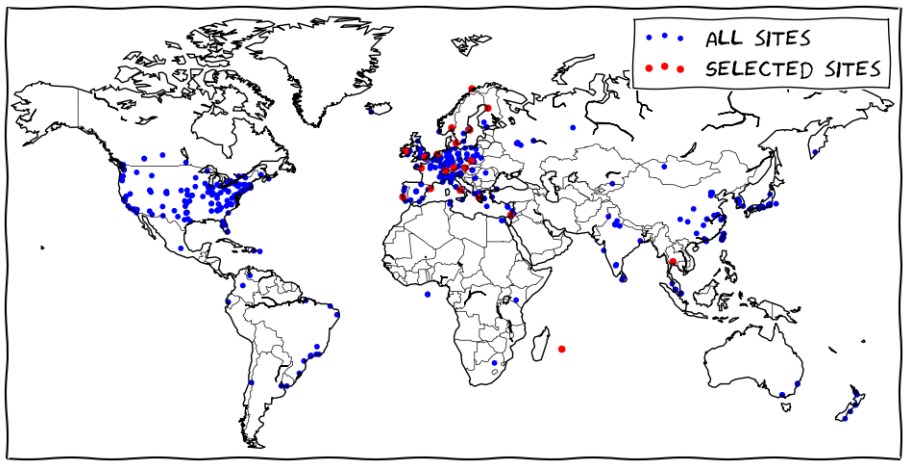

Figure 5.3: Geographical distribution of PlanetLab sites

Since we want to avoid stressing the live Tor network with too many concurrent measurements, we restrict the number of hosts that will run our measurements. We aimed to select hosts that on the one hand are distributed across different countries and ISPs, and on the other hand provide sufficient spare resources in terms of computational time, memory, and bandwidth.

Of the 1041 worldwide available nodes on PlanetLab, we had access to 223 nodes from PlanetLab Europe. From those nodes we had access to, we selected 19 nodes, biasing towards diversity in terms of geographical distribution, upstream ISP, and country where possible. Additionally we aimed for nodes which have plenty spare hardware resources and were operated on different sites.

In slight deviation from the default PlanetLab environment, Fedora Linux 14, we had to install the following additional software packages from more recent Fedora's repositories: python-lzo, and stem. To meet the requirements of our measurement software, we also had to recompile two software packages, curl and libcurl, to run on Fedora 14. After that, we only had to copy a statically compiled, patched Tor binary and our measurement software to the hosts.

5.5 Measurement and Analysis Methods

To examine the correlation between both RTTs and TTFBs, and CBTs and TTFBs, and to evaluate the influence of the Circuit-RTT approach on bandwidth and anonymity, we conduct direct measurements of circuits on the live Tor network. We are especially interested in RTT measurements over time for individual circuits and the distribution of RTT values over many circuits. Additionally, we examine the influence of the use of Guard nodes on latency.

RTT Measurement

After a circuit is built, we ask the exit node to open a TCP connection to an IP address from the 127.0.0.0/8 network. We use socksiPy, a Python library for SOCKS, to connect our TCP stream

to the Tor client.[7] Since the exit policy always refuses connections to the local network, the exit node returns an error. We define the time between sending the connection request and receiving the reply as the RTT of a circuit, since we assume that it is very close to the real RTT of a circuit.

TTFB Measurement

We approximate user-experienced network latency with the time elapsed between sending a HTTP request through the Tor network and receiving the first byte of its response, the so-called TTFB. To measure the TTFB, we request the "google.com" website using the HTTP HEAD method. This method requests a response similar to the HTTP GET method, but HTTP HEAD requests headers only without the content body. We chose google.com as destination, because it introduces only a small bias towards the exit node's position on the Internet, as it is served from multiple data centers across the globe, such that it provides fast response times from all over the world.

Measurements were conducted using PycURL, a Python interface to libcURL, a library for transferring data using various protocols, including HTTP. We define the TTFB as the difference between cURL's *"STARTTIME"* and *"CONNECTTIME"*, where *"STARTTIME"* denotes the time from the start until the first byte is received and *"CONNECTTIME"* denotes the time from the start until the connection to the SOCKS proxy[8] has been established.

Bandwidth Measurement

To evaluate any possible influence the Circuit-RTT method has on bandwidth, we measure the achieved throughput through Tor circuits, too. To that end, we registered the domain *torrtt.info* and configured a webserver to serve a static HTML (HyperText Markup Language) page, which contains generated lorem ipsum text.[9] The size of that web page is exactly 5 MB, in accordance with the size used by Tor's bandwidth scanners for their bandwidth measurements.

To reduce the bias a single webserver serving the web page could introduce, the page is delivered by CloudFlare's worldwide distributed cluster. CloudFlare is a Content Delivery Network (CDN) that aims to improve website performance. We use it to reduce the distance and thus, potentially, the latency between the exit node of a particular circuit and the destination webserver. In slight deviation from CloudFlare's default settings, we also configured CloudFlare to cache our static HTML page.

We again use PycURL as HTTP client. The 5 MB HTML file from our test website[10] is requested with the HTTP header *"Accept-Encoding"* set to *"identity"*, so that the reply is served uncompressed. We define the time it takes to transfer 5 MB as the difference between cURL's *"TOTALTIME"* and *"STARTTRANSFERTIME"*, where *TOTALTIME* is the total time for the transfer, including name resolving, TCP connect etc., and *STARTTRANSFERTIME* is the time it takes from the start until the first byte is received.

[7] Since the socksiPy library is unmaintained, we had to patch a small, known programming error.

[8] From cURL's point of view the Tor client is the SOCKS proxy.

[9] We implemented the bandwidth measurements after the TTFB measurements and concluded that we could not use Google's website for those measurements.

[10] `http://www.torrtt.info/index.html`

We verified that the answers are indeed uncompressed by examining the HTTP headers served by CloudFlare's caching webservers.

Anonymity Metrics

Before performance-optimizing modifications can be integrated into Tor, it is important to study their impact on anonymity. In low-latency systems, anonymity is often measured as the probability that an adversary occupies both the entry and the exit node of the user's circuit.

Shannon Entropy

Serjantov and Danezis [23] and Diaz et al. [24] independently proposed evaluation frameworks, that use Shannon entropy computed over a set of potential senders as metric for quantifying anonymity. Calculating Shannon-Entropy over the distribution of entry and exit node combinations helps us to quantify the bias in node selection and increases the comparability of our measurement results. We calculate the entropy over all those combinations Ψ by

$$E(X) = -\sum_{i=1}^{|\Psi|} p_i * ld(p_i)$$

i refers to a particular entry/exit-combination and p_i to the probability that the combination i be used in a circuit. The entropy is then normalized by calculating

$$E_{norm} = \frac{E(X)}{E_{max}}$$

E_{max} refers to the maximum possible entropy $ld(|\Psi|)$. Normalizing the entropy relative to the maximal entropy ensures that $E_{norm} \in [0, 1]$, where $E_{norm} = 1$ implies that nodes are selected uniformly at random, and $E_{norm} < 1$ implies a bias.

Gini Coefficient

Another metric for quantifying anonymity calculates how many nodes an attacker must subvert in order to compromise a circuit. Skewing node selection towards certain nodes decreases the number of nodes an attacker must compromise, while choosing nodes uniformly increases it. To quantify such inequality of node selection, Snader and Borisov propose to adopt the Gini coefficient, an equality metric commonly used in economics. It is defined as

$$G = \frac{1}{\mu} \int_0^\infty CDF(x)(1 - CDF(x))\,\mathrm{dx}$$

CDF is the observed Cumulative Distribution Function of node selection and μ is its mean. A Gini coefficient $G = 0$ represents perfectly uniform node selection, and $G = 1$ implies perfect inequality, i.e. the same node is always chosen.

CHAPTER 6

Results

> *In theory there is no difference between theory and practice. In practice there is.*
>
> Unknown author

As part of this study, we observe and analyze the influence of the CBT and the Circuit-RTT methods on the quality of circuits with regard to latency, bandwidth, and anonymity. To that end, we have conducted several experiments, whose results will be presented in this Chapter. The basic idea of the Circuit-RTT approach is that clients create a local view of achievable latency by actively measuring the circuits' RTTs. After a circuit has been established, the client measures the circuit's RTT and drops the circuit if its RTT is above a certain, previously calculated cut-off value. This ensures that a user gets a fast, pre-built circuit when it is needed.

6.1 RTT Measurements on Individual Circuits

In the first experiment we conducted, RTTs of individual circuits were measured on the live Tor network. Running 50 threads concurrently on a single host computer, circuits were established over a few days and the RTT on each circuit was measured up to 100 000 times. Using the method described in Section 5.1 on page 36, we examined the stability of RTT measurements over time.

Results

Figure 6.1 on the next page depicts the probability distributions of RTTs observed on four different circuits, representative of the results we got from this experiment. The circuits seem to

be subject to various influences, since RTTs vary a lot on individual circuits. The measurements are far from being constant and it seems impossible to determine a constant RTT of a circuit.

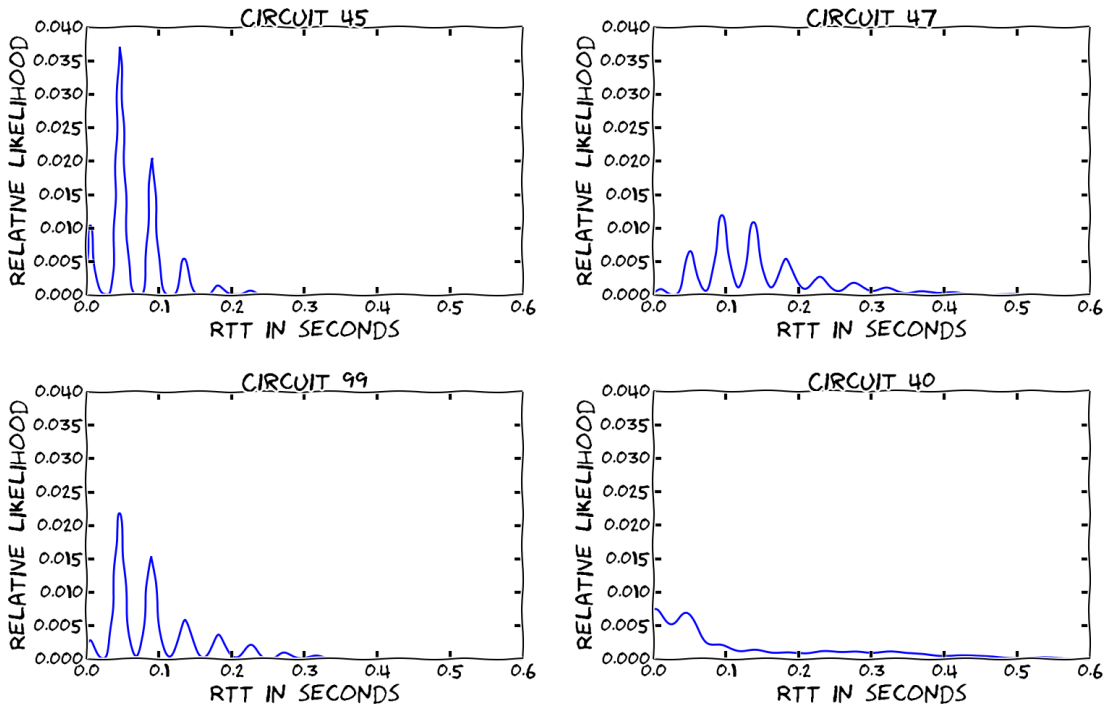

Figure 6.1: Probability distribution of RTTs on individual circuits.

Even though one shape often emerges, no single statistical distribution seems to fit the RTT measurements of every circuit. The occurrence of other shapes does not seem necessarily to imply high RTTs. We did explore some methods to estimate the quality of a circuit. However, we could not find one that both is generally applicable and significantly improves latency. Occasionally we identified spikes at 100 ms or 1000 ms intervals. Since these values correspond to the token bucket refill intervals previously described in Section 3.3 on page 20, we assume that some method could be used to bias circuit selection towards high bandwidth circuits. However, since this would be hard to analyze, and our focus lies on improving latency rather than throughput, we consider this to be outside of the scope of our thesis and future work.

When our measurement software creates a new TCP connection, we attach it to the appropriate circuit, so that the connection is tunneled through that circuit. For the majority of circuits this procedure of attaching TCP streams failed at some points over time. No relation to nodes' stable flags could be determined, although we did not investigate reasons for stream attachment failures in great detail.

Conclusion

Defining the quality of a circuit, and so suggesting the most appropriate metric for assessing circuits, highly depends on the application. For some applications, such as VoIP, worst-case optimization would be appropriate, but, in general, average-case optimization is more suitable. However, a lot of measurements would be required and the load on Tor network would be increased tremendously, if every client used per-circuit statistics. For applications requiring real-time service, such as VoIP, the increased cost of multiple measurements and the potential compromise of anonymity by hard cut-off limits[1] may be justified. We could not find a single approach that optimizes both the average and the worst-case RTT. Regardless of the number of measurements, sudden changes in network conditions cannot be predicted.

For this reasons, as we aim for general applicability, we focus on a different approach. Although the variance of circuit latency over time is considerably high, we do not completely dismiss the basic technique of measuring RTTs. Instead of further examining approaches that would require tens or hundreds of RTT measurements on each circuit, we will focus on doing a single measurement per circuit, considering the overall distribution alone - similar to the CBT method, where only one measurement value is considered. A circuit's RTT is probed once right after construction and the overall distribution of RTTs is then used to select a timeout value across all circuits.

6.2 Single RTT Measurements on Various Circuits

With the next experiment, we aim to examine the overall distribution of circuits' RTTs, assuming that they are - apart from the parameters - identical across clients, and can also be approximated by a Fréchet distribution, like the CBTs. This experiment was also conducted on the live Tor network. To simulate clients on different networks, we have chosen 19 hosts from PlanetLab, as described in Section 5.4 on page 42. Each client built up to one million circuits, measuring the RTT once on each circuit, using ten concurrently running threads. The experiment took about 14 days to complete.

Results

Figure 6.2 on the following page depicts the PDFs of RTTs, each from one million circuits and the corresponding GEV distribution with estimated parameters. The GEV distribution is a superset of extreme value distributions, including the Fréchet distribution.[2] There is a very practical reason for using GEV over Fréchet: For estimating the distribution's parameters, we used *"R"*, a programming language and software environment for statistical computing. We thus relied on the availability of suitable software packages. In R such package only exists for the

[1] Such as "RTT $\leq$ 500ms"

[2] As explained in Section 3.3 on page 20, it was empirically found that CBTs fit a Fréchet distribution. Estimators for that distribution converge slowly and are difficult to calculate and thus instead of calculating estimators for a Fréchet distribution its tail is approximated with a Pareto distribution. [cf. 43]

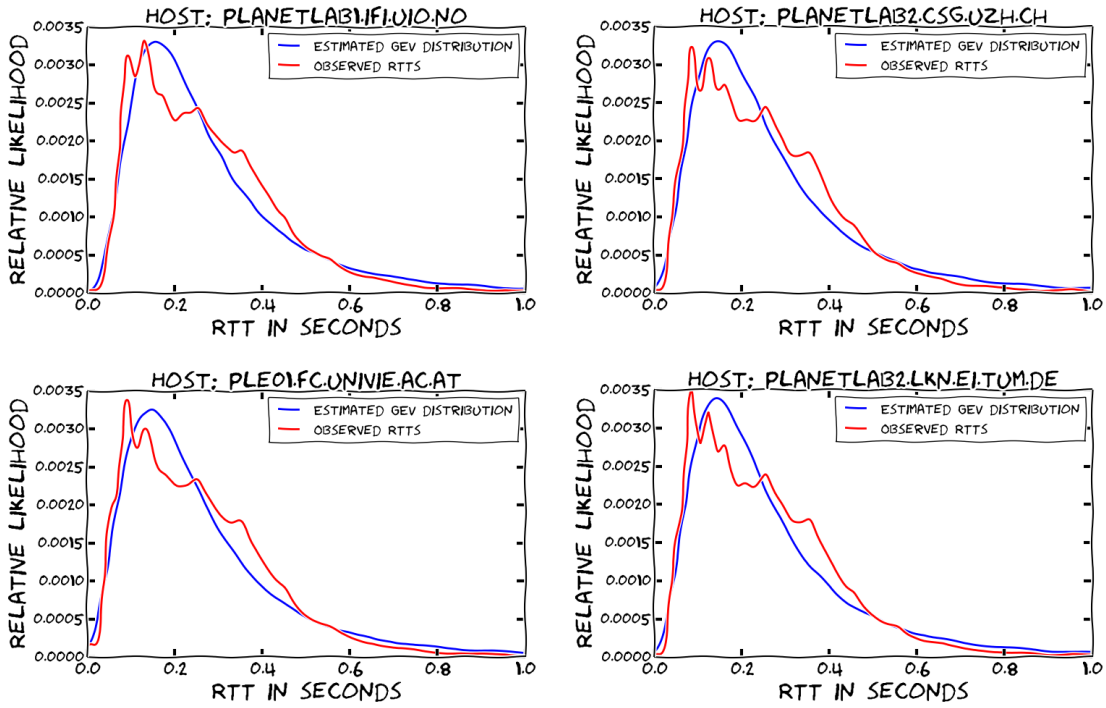

Figure 6.2: PDF of one million observed RTTs and the corresponding GEV distributions on various hosts.

GEV but not for the Fréchet distribution. Since GEV is a superset of extreme value distributions also including Fréchet, it is possible to use GEV and still have applicability for Fréchet.[3]

These results support our assumption that the probability distribution of RTTs is identical across different clients, ignoring the differences resulting from the particular distribution parameters. Compared to previous research, our results show that an improvement of the median RTT from 0.86s [see 21] or 0.6s [see 16, p. 68][4] to 0.27s. Table 6.1 on page 53 lists more detailed results. As the performance in the Tor network improved tremendously within the last years, we are confident to have gathered valid results.

Figure 6.3 on the next page depicts the RTTs of one million circuits over time. We note that the RTT values are unstable over time and that a phasing pattern can be identified.[5] We can clearly identify a day/night pattern, where latency is lower, with less deviation during night hours than during the day time, when latency is more scattered. This could be caused by the local network or the upstream ISP of the particular client, or the Tor network itself. Although we did not investigate these patterns in detail, we believe that the most plausible explanation is that these patterns are caused by the unequal distribution of Tor nodes and clients around the globe.

[3]GEV is equivalent to Fréchet, as long as the parameter $\xi > 0$, which it is in this case.

[4]Including additional latency from the exit node to the destination server.

[5]These phases can also be identified for CBT and TTFB values.

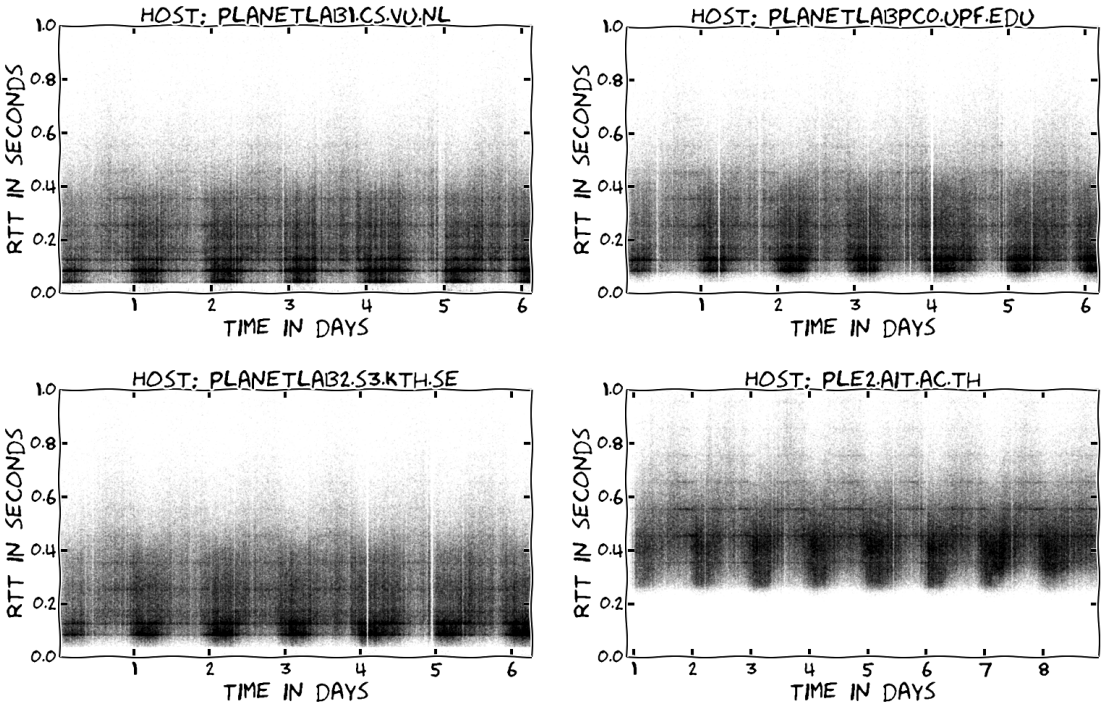

Figure 6.3: RTT over time showing day/night patterns on different hosts.

6.3 Measuring RTTs, CBTs, and TTFBs on Various Circuits

In the experiment described in Section 6.2 on page 49, we also measured the circuits' build times and the TTFBs. We approximate network latency as experienced by users with the time elapsed between sending a HTTP request to the google.com website through the Tor network and receiving the first byte of its response - the so-called TTFB, as explained in Section 5.5 on page 44. Furthermore, we extract the CBT values from Tor's general log messages, as explained in Section 5.4 on page 42.

Results

Figure 6.4 on the next page depicts the probability distributions of one million CBTs on four PlanetLab hosts. The shape of the PDFs across all hosts can be approximated well by a GEV distribution. These results confirm that CBTs can indeed be approximated by a Fréchet distribution. We measured a median CBT of about 0.84s, which again shows the improvement of the Tor network's performance in recent years.[6]

Table 6.1 on page 53 shows the median values for the observed CBT, RTT, and TTFB values in on different PlanetLab hosts. In comparison to data from other researchers, we see an improvement of latency from 0.6s in 2010 [see 16, p. 68] to about 0.34s in our measurements

[6]Previous research measured a median of 2.11 s. [see 21])

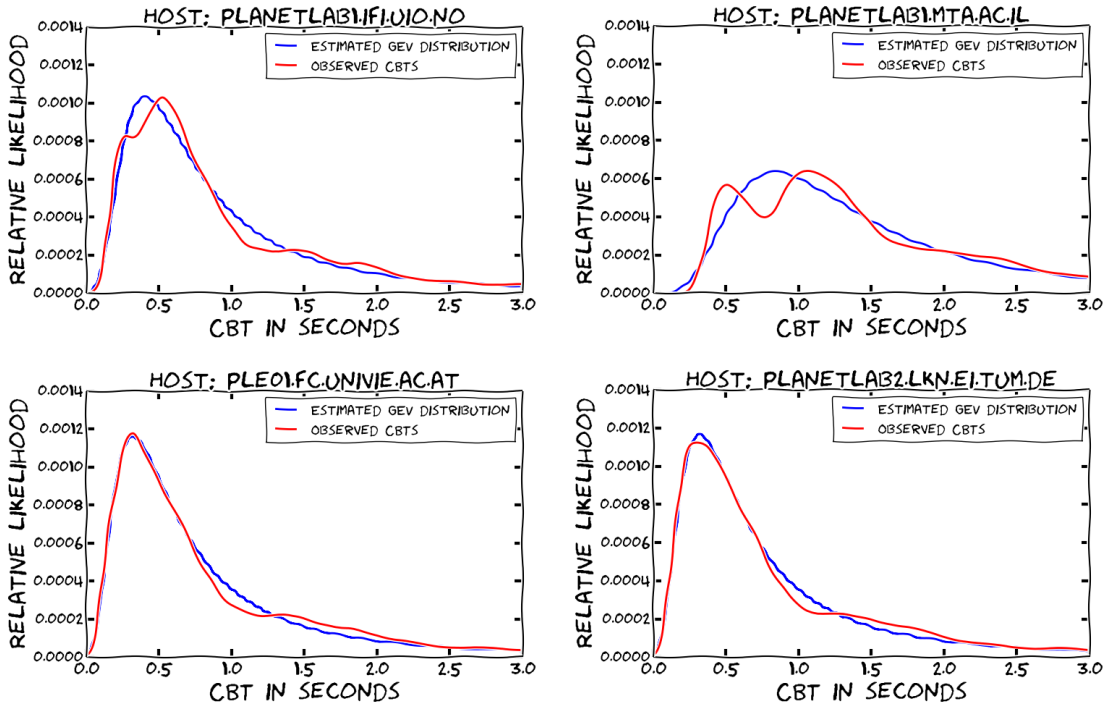

Figure 6.4: PDFs of one million observed CBTs and corresponding GEV distributions.

from 2014. The difference can be explained on the one hand by the improvement of the Tor network and on the other hand by our attempt to get very fast response times from Google's web servers.

Correlation

To determine which of the CBT or the Circuit-RTT methods is more suitable to estimating latency, we examine the correlation both between CBT and TTFB and between Circuit-RTT and TTFB.

For this purpose, we have to select an appropriate statistical method. A widely used measure of the linear correlation of two variables is the Pearson product moment correlation coefficient (Pearson's r). Since it is not robust if outliers are present [see 67], we use scatterplots to examine the existence of such outliers. In the scatterplots on pages 54 and 55, data is displayed as a set of points, each with the CBT or RTT on the x-axis and the corresponding TTFB values on the y-axis. These plots suggest a positive correlation both between CBT and TTFB and between RTT and TTFB. They further indicate a stronger correlation between RTT and TTFB. It is important to note that we can indeed identify many outliers. For this reason, we can not make use of Pearson's r. Instead, we use Spearman's ρ, which is resistant to the existence of outliers because values are converted to ranks. Spearman's rank correlation coefficient is a nonparametric measure of statistical dependance between two variables. It is essentially Pearson's r on the ranked

PlanetLab Host	Median CBT	Median RTT	Median TTFB
planetlab2.cs.uit.no	830 ms	263 ms	340 ms
pl001.ece.upatras.gr	861 ms	266 ms	342 ms
planetlab2.s3.kth.se	749 ms	247 ms	322 ms
planetlabpc0.upf.edu	789 ms	257 ms	332 ms
planetlab-tea.ait.ie	676 ms	241 ms	316 ms
planetlab1.cs.vu.nl	577 ms	226 ms	300 ms
ple2.ait.ac.th	2402 ms	461 ms	539 ms
planetlab2.virtues.fi	763 ms	255 ms	331 ms
planetlab-3.iscte.pt	809 ms	260 ms	336 ms
planetlab1.mta.ac.il	1284 ms	336 ms	412 ms
planetlab2.csg.uzh.ch	644 ms	237 ms	312 ms
planetlab1.diku.dk	624 ms	233 ms	307 ms
plab4.ple.silweb.pl	730 ms	250 ms	327 ms
planetlab1.nrl.eecs.qmul.ac.uk	569 ms	223 ms	299 ms
planetlab2.lkn.ei.tum.de	618 ms	232 ms	306 ms
ple01.fc.univie.ac.at	624 ms	232 ms	308 ms
planetlab01.dis.unina.it	833 ms	271 ms	345 ms
planetlab1.ifi.uio.no	714 ms	247 ms	321 ms
planetlab1.jcp-consult.net	878 ms	269 ms	348 ms

Table 6.1: CBT, RTT, and TTFB values measured on 19 PlanetLab hosts.

values rather than on the observed values, and ranges from -1, indicating a negative correlation, to 1, indicating a positive correlation. A value of 0 suggests that there is no linear correlation.

Table 6.2 on page 56 shows Spearman's correlation coefficient ρ for CBT and RTT on all measurement hosts. Furthermore it lists the number of successful measurements on each host.[7] We note a positive correlation with TTFB for both types of values consistently over all measurement hosts. However, the correlation coefficient is significantly higher for RTTs than for CBTs. The two-sided p-value for the hypothesis test, whose null hypothesis is that two sets of data are uncorrelated, is 0 for all measurements. This suggests a very strong presumption against the null hypothesis, and thus confirms a correlation between CBT and TTFB and between RTT and TTFB.

As we now have evidence that there is a very high correlation between RTT and TTFB, we assume that the Circuit-RTT method can be used further to improve latency in the Tor network.

6.4 Calculating Cut-Off Values

The key property of both the CBT and the Circuit-RTT method is that they are adaptive. A Tor client will not use every circuit, but only a certain percentage of them. CBT currently uses the

[7] We experienced reliability issues on some hosts, resulting on a lower total number of measurements. We consider the measurement results very conclusive, nonetheless.

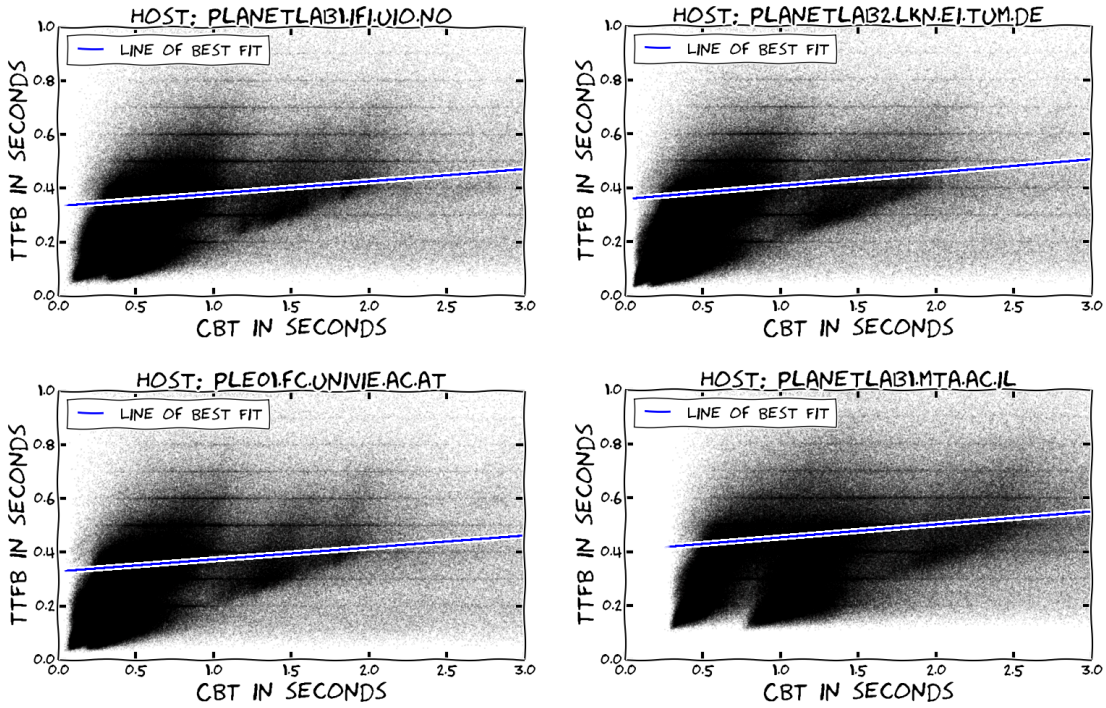

Figure 6.5: Scatterplots visualizing the relation between CBTs and TTFBs.

fastest 80% of all circuits by default, so that a client rejects 20% of all circuits before they can be used. It is important to verify that the percentage of circuits that would be rejected by a client does not vary between PlanetLab hosts. The specific cut-off values may vary, but the overall percentage of pruned circuits must not.

Figure 6.7 on the facing page depicts the basic algorithm we use to calculate the percentile of a specific CBT or RTT value and so determine on whether that circuit would have been used or not.[8] The maximum number of measurements used to estimate the GEV distribution's parameters is 1000, in accordance with the current CBT implementation, which makes a reasonable estimation possible. We hold 1000 RTT values in a circular array and estimate the parameters for the corresponding GEV-distribution. With these parameters, we calculate the corresponding GEV curve and then the percentile for the new RTT value.

In Figure 6.8 on page 57 the comparison of cut-off limit settings to the effective cut-off limits for CBT and Circuit-RTT on different hosts is depicted. A line of best fit is also drawn for reference. We note that the statistical prediction works reasonable well for both CBT and Circuit-RTT, as the effective cut-off limits are close to the set limits.

[8] If the estimation of GEV parameters failed, we use the last utilizable data as replacement.

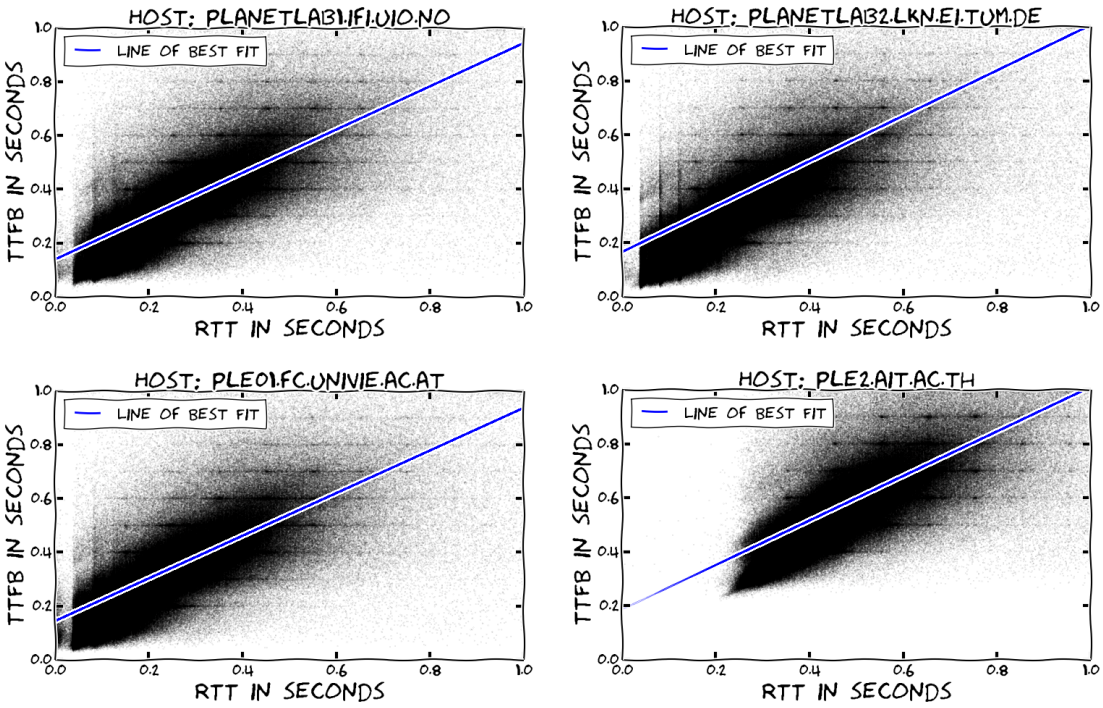

Figure 6.6: Scatterplots visualizing the relation between RTTs and TTFBs.

Algorithm 6.4.1: CALCULATE PERCENTILE()

FILL CIRCULAR ARRAY WITH 1000 MEASUREMENT VALUES()
while MEASUREMENT VALUES LEFT()
do
- ADD NEW MEASUREMENT VALUE TO CIRCULAR ARRAY()
- ESTIMATE GEV DISTRIBUTION PARAMETERS()
- CALCULATE PERCENTILE OF MEASUREMENT VALUE()
- SAVE MEASUREMENT INCLUDING THE CALCULATED PERCENTILE()

Figure 6.7: Percentile calculation

PlanetLab Host	ρ (CBT)	ρ (RTT)	# of Measurements
planetlab2.cs.uit.no	0.5638	0.8247	949 274
pl001.ece.upatras.gr	0.5669	0.8288	863 477
planetlab2.s3.kth.se	0.5463	0.8245	948 970
planetlabpc0.upf.edu	0.5732	0.8245	949 672
planetlab-tea.ait.ie	0.5963	0.8201	949 428
planetlab1.cs.vu.nl	0.6302	0.8131	949 706
ple2.ait.ac.th	0.3498	0.7918	947 230
planetlab2.virtues.fi	0.5864	0.8274	950 137
planetlab-3.iscte.pt	0.5884	0.8251	190 439
planetlab1.mta.ac.il	0.4762	0.7863	949 034
planetlab2.csg.uzh.ch	0.6175	0.8277	765 987
planetlab1.diku.dk	0.6372	0.8340	384 313
plab4.ple.silweb.pl	0.5905	0.8117	950 491
planetlab1.nrl.eecs.qmul.ac.uk	0.6435	0.8331	383 938
planetlab2.lkn.ei.tum.de	0.6254	0.8286	951 048
ple01.fc.univie.ac.at	0.6257	0.8266	949 786
planetlab01.dis.unina.it	0.5698	0.7215	479 199
planetlab1.ifi.uio.no	0.5939	0.8246	950 546
planetlab1.jcp-consult.net	0.5410	0.8002	628 088

Table 6.2: Spearman's ρ for CBT and RTT on all measurement hosts.

6.5 Latency Evaluation

In this Section, we will verify our assumption that Circuit-RTT can be used further to improve latency in the Tor network. Using the data gathered from the experiment described in Section 6.2 on page 49, we will explore the effect of different cut-off percentages used with CBT, Circuit-RTT, and a combination of both methods on TTFB.

Results

Figure 6.9 on page 58 depicts the average TTFB for all possible cut-off limit settings and an optimum TTFB line for reference. We note that Circuit-RTT approaches that ideal line, with only minimal effort. As assumed, the average TTFB improves when we reduce the percentage of accepted circuits for both methods CBT and Circuit-RTT.

Table 6.3 on page 58 shows the average and the 90th percentile TTFB values using the CBT and Circuit-RTT methods with certain cut-off settings.[9] When using the fastest 80% of circuits, the average TTFB is reduced from about 0.40s by 17.4% to 0.33s for CBT and by 22.1% to

[9] Errors during calculating the statistical data for three hosts (planetlab2.s3.kth.se, ple2.ait.ac.th, and planetlab-3.iscte.pt) were encountered. Hence, those hosts are not listed in this table. Nevertheless, we find the data conclusive as is.

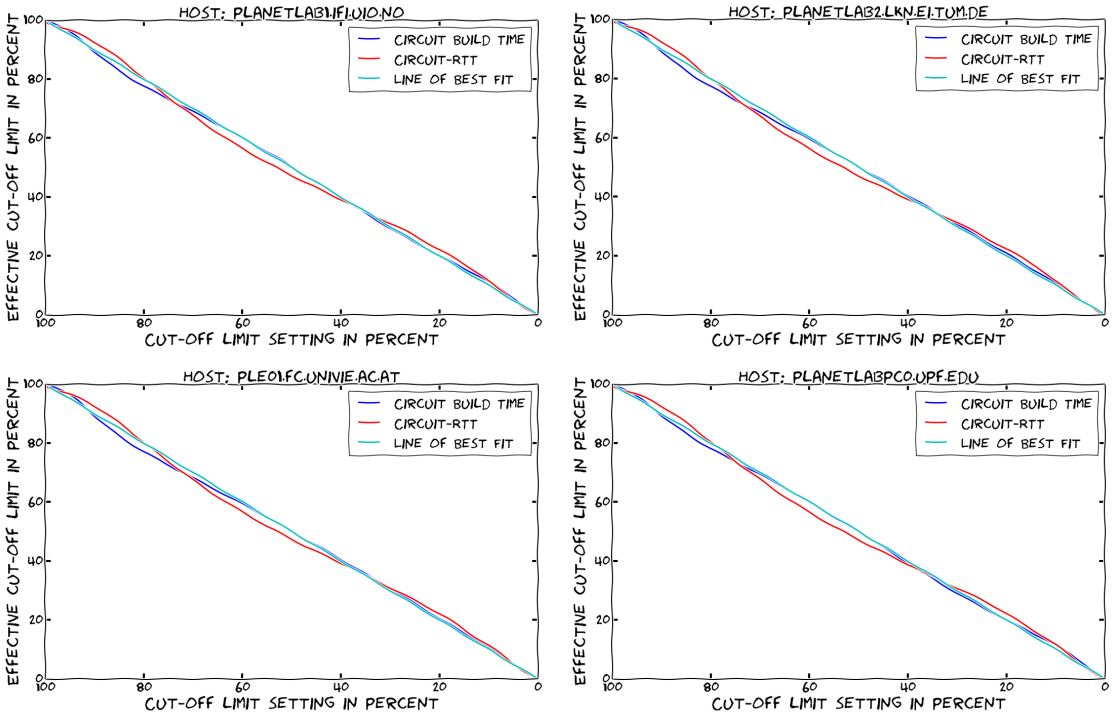

Figure 6.8: Comparison of cut-off limit settings to effective cut-off limits for CBT and Circuit-RTT.

0.31s for Circuit-RTT. Using the fastest 60% of circuits improves the TTFB even more, as the average TTFB is reduced by 25.6% to 0.3s for CBT and by 35% to 0.26s for Circuit-RTT.

But the average TTFB is less improved in absolute numbers than the 90th percentile, which represents the maximum latency for 90% of observered circuits. When using the fastest 80% of circuits, the 90th percentile TTFB is reduced from 0.61s by 13% to 0.53s for CBT and by 20.3% to 0.48s for Circuit-RTT. Reducing the percentage of accepted circuits to 60% improves the TTFB even further, where the 90th percentile TTFB is reduced by 21.4% to 0.48s for CBT and by 33.4% to 0.40s for Circuit-RTT.

We note that Circuit-RTT has more of an effect on the TTFB than CBT, for any cut-off limit except very low ones, at which hardly any circuit would be used anymore. We find that, Circuit-RTT improves latency more than CBT, the currently used method, does. But the question of whether a combination of both methods could improve latency even further remains.

Combination of CBT and Circuit-RTT To explore a combination of both approaches, we only consider circuits that meet the cut-off criteria for both CBT and Circuit-RTT methods at the same time, and then calculate the effective cut-off percentage to make the results comparable.

Figure 6.10 on page 59 depicts the average TTFB for the effective cut-off limits using the CBT, the Circuit-RTT, and a combination of both methods. We note that Circuit-RTT improves TTFB more than either CBT or the combination of both methods does. Hence, we will focus on

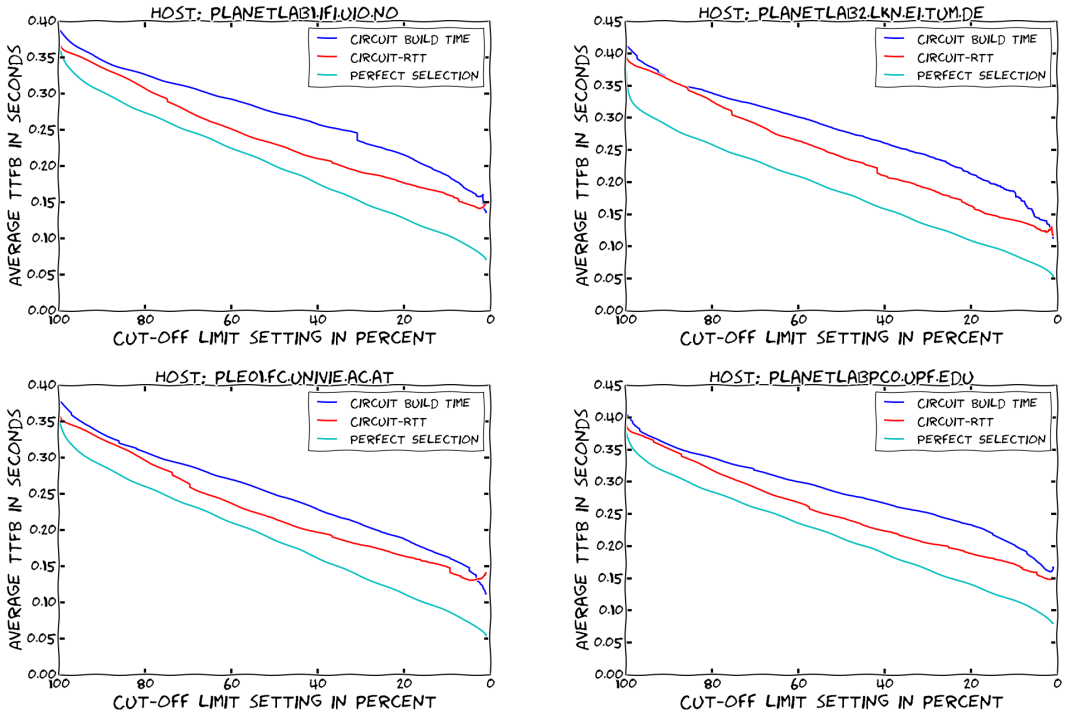

Figure 6.9: Average TTFB for different CBT and Circuit-RTT cut-off limits.

PlanetLab Host	CBT(80)	Circuit-RTT(80)	CBT(60)	Circuit-RTT(60)
planetlab2.cs.uit.no	340/539	329/495	305/494	266/408
pl001.ece.upatras.gr	340/536	324/494	308/492	269/409
planetlabpc0.upf.edu	336/529	318/487	300/483	266/402
planetlab-tea.ait.ie	319/513	283/461	306/472	255/393
planetlab1.cs.vu.nl	299/500	260/435	282/457	229/376
planetlab2.virtues.fi	333/529	315/488	299/480	260/402
planetlab1.mta.ac.il	425/642	403/594	396/600	348/503
planetlab2.csg.uzh.ch	312/511	295/470	274/455	240/388
planetlab1.diku.dk	308/500	294/460	270/439	238/381
plab4.ple.silweb.pl	329/529	314/490	293/478	260/403
planetlab1.nrl.eecs.qmul.ac.uk	298/491	281/445	258/425	229/368
planetlab2.lkn.ei.tum.de	337/503	324/461	301/446	263/382
ple01.fc.univie.ac.at	307/504	296/464	269/446	236/383
planetlab01.dis.unina.it	347/547	338/517	313/498	291/447
planetlab1.ifi.uio.no	325/517	307/477	292/468	250/397
planetlab1.jcp-consult.net	379/559	354/507	339/505	294/426

Table 6.3: Median and 90th percentile TTFB values for CBT and Circuit-RTT using certain cut-off settings.

Circuit-RTT as a replacement for CBT, instead of using it as an additional method. CBT might be used in addition to Circuit-RTT to avoid CPU-congested nodes, but not to improve latency further.

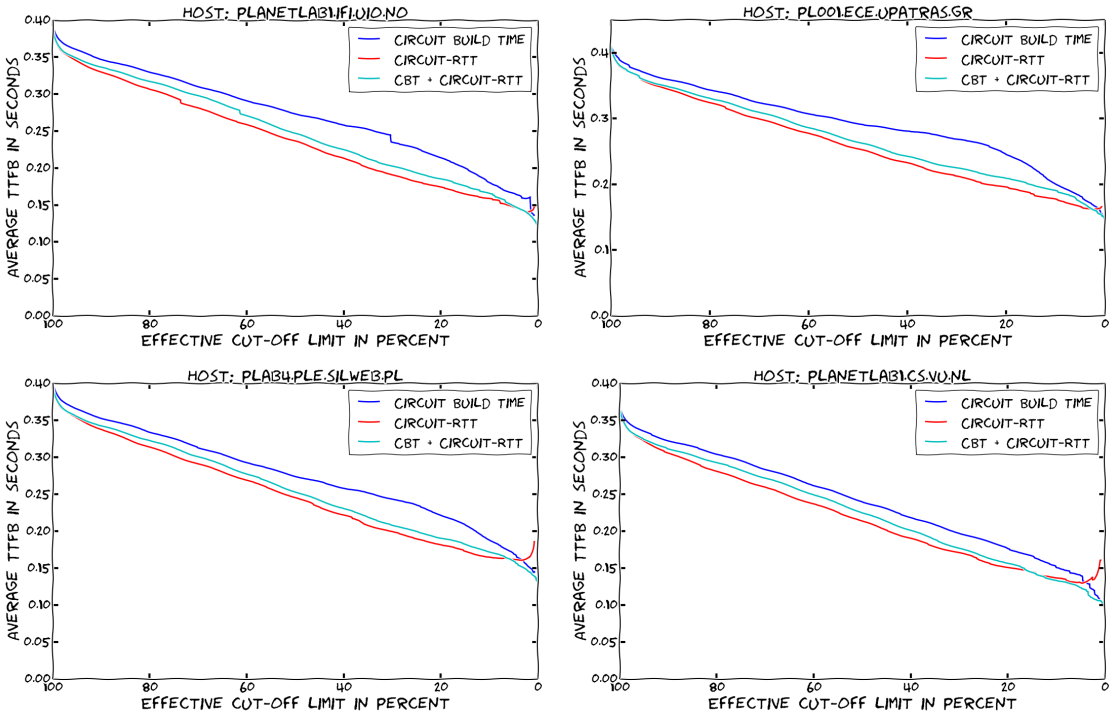

Figure 6.10: Average TTFB for the effective cut-off limits of CBT, of Circuit-RTT, and of their combination.

6.6 Bandwidth Evaluation

In addition to examining the effect of the use of CBT and Circuit-RTT on latency as experienced by users, we also study the effect those methods have on achievable throughput, which is also a property important for users. Assuming that the use of both methods CBT and Circuit-RTT leave the achieved throughput unaffected, we conducted another experiment on seven hosts from PlanetLab over about three weeks. Each host built up to 100 000 circuits and measured the achieved throughput. This time, only three concurrently running threads were used to ensure that the limited bandwidth available on each measuring host did not have any negative influence on the results. We measure the achieved throughput over circuits by downloading a single 5 MB HTML file from CloudFlare's CDN, as described in Section 5.5 on page 44.

Figure 6.11 on the next page depicts the average bandwidth observed for different CBT or Circuit-RTT cut-off limits on four PlanetLab hosts. Contrary to our assumption that throughput is affected by the use of neither the CBT nor the Circuit-RTT method, the observed bandwidth

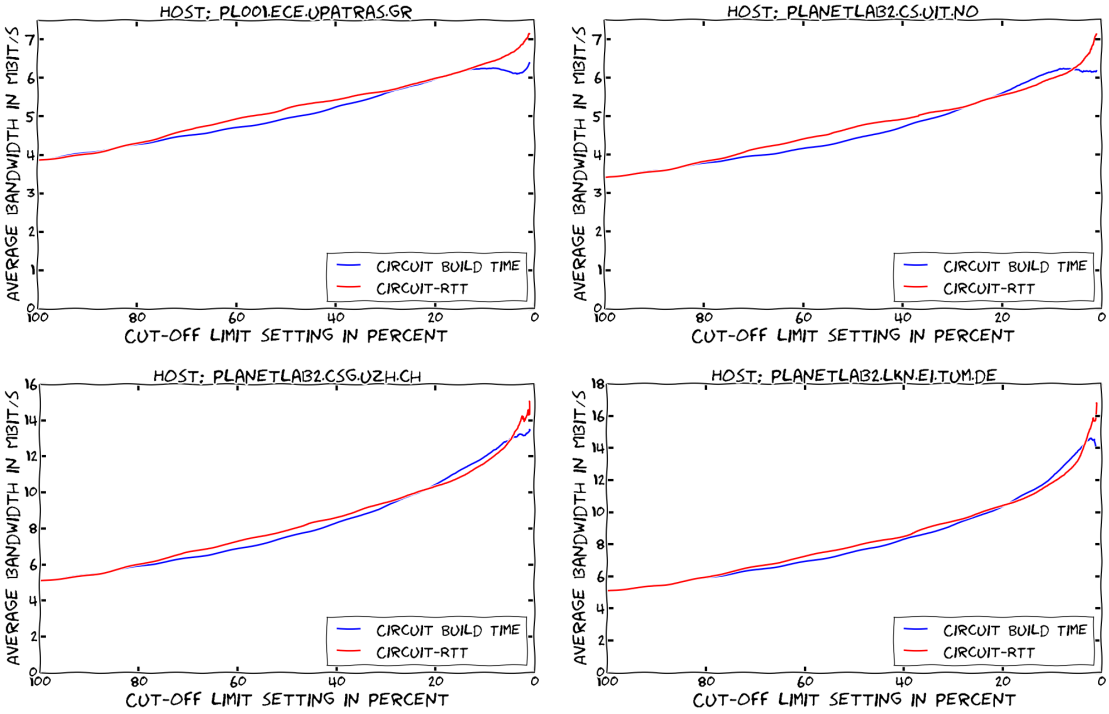

Figure 6.11: Average bandwidth with CBT and Circuit-RTT using different cut-off limit settings.

PlanetLab Host	CBT(80)	Circuit-RTT(80)	CBT(60)	Circuit-RTT(60)
pl001.ece.upatras.gr	2.58/1.35	2.63/1.41	2.97/1.57	3.19/1.73
planetlab2.cs.uit.no	2.09/1.08	2.14/1.13	2.39/1.20	2.58/1.29
planetlab2.csg.uzh.ch	2.47/1.18	2.56/1.21	3.05/1.38	3.18/1.44
planetlab2.lkn.ei.tum.de	2.44/1.17	2.50/1.20	3.00/1.35	3.13/1.42
planetlab2.s3.kth.se	2.07/1.12	2.13/1.16	2.35/1.23	2.52/1.35
planetlab2.virtues.fi	2.28/1.15	2.35/1.19	2.67/1.27	2.92/1.43
ple2.ait.ac.th	1.11/0.81	1.14/0.85	1.16/0.88	1.20/0.93

Table 6.4: Average and 90th percentile bandwidth with CBT and Circuit-RTT using certain cut-off settings.

increased when less circuits were used using any of both methods. For reasonable cut-off limits, Circuit-RTT performs slightly better than CBT. The bandwidth values we measured are in line with those on the Tor metrics website [see 68], confirming the accuracy of the data we gathered in our experiment.

Table 6.4 shows the average and the 90th percentile bandwidth values using the CBT and Circuit-RTT methods with certain cut-off settings. When using the fastest 80% of circuits, the average bandwidth is increased from 1.83Mbit/s by 19% to 2.26Mbit/s for CBT and by 20.8%

to 2.31Mbit/s for Circuit-RTT. Using the fastest 60% of circuits increases the bandwidth even further, with the average bandwidth increased by 31% to 2.65Mbit/s for CBT and by 36.2% to 2.87Mbit/s for Circuit-RTT.

6.7 Anonymity Evaluation

Before performance-optimizing modifications can be integrated into Tor, it is important to study their effects on the anonymity the system provides. In low-latency systems, anonymity is usually approximated by the probability that an adversary occupies both the entry and the exit node of the user's circuit. With this in mind, we quantify anonymity to estimate the strength of protection that would have been achieved for end-users from our earlier experiments on 19 hosts that tested one million circuits each. See Section 6.2 on page 49 for details of the experimental setup. We study the trade-off between the quality of protection and the quality of service, assuming that anonymity is slightly decreased, but that latency improvements outweigh this reduction in anonymity. As explained in Section 5.5 on page 46, we use normalized Shannon entropy over the observed distribution of entry/exit-node combinations to quantify anonymity. $E_{norm} = 1$ implies that nodes are selected uniformly at random, and $E_{norm} < 1$ implies a bias. We also use the Gini coefficient as an anonymity metric, to quantify the skew in the selection of entry and exit nodes towards certain nodes. A Gini coefficient $G = 0$ represents perfectly uniform node selection, and $G = 1$ implies perfect inequality, i.e. one particular node is always chosen.

Results

Table 6.5 on the next page shows the Gini coefficient for both CBT and Circuit-RTT methods with certain cut-off settings.[10] When using the fastest 80% of circuits, the Gini coefficient is worsened from 0.2075 overall by 7.9% to 0.2252 for CBT and by 8.4% to 0.2265 for Circuit-RTT. Using the fastest 60% of circuits further increases the bias towards certain nodes. The Gini coefficient is increased by 8.7% to 0.2271 for CBT and by 17.5% to 0.2516 for Circuit-RTT. We note that there is slightly more bias towards certain nodes for Circuit-RTT than for CBT at the default cut-off setting. This bias increases when the cut-off setting is reduced.

Table 6.6 on the next page shows the normalized Shannon entropy values for both CBT and Circuit-RTT methods. When using the fastest 80% of circuits, the normalized Shannon entropy is decreased from 0.8403 overall by 21.8% to 0.6568 for CBT and by 18.6% to 0.6844 for Circuit-RTT. Using the fastest 60% of circuits further decreases entropy. The normalized entropy is decreased by 39.5% to 0.5083 for CBT and by 42.1% to 0.4862 for Circuit-RTT. Unlike the Gini coefficient, entropy is improved by Circuit-RTT compared to the values for CBT at the default cut-off limit.

Figure 6.12 on page 63 shows that normalized Shannon entropies hardly varies both between the CBT and the Circuit-RTT method when using the same cut-off limits. Depending on the cut-off setting, one method is often slightly better than the other. We get the same value (0.83) as previous research [see 21], which further confirms the correctness of our results. In contrast to the study cited, Circuit-RTT achieves more improvement in latency without requiring a hard

[10] See footnote 9 on page 56.

PlanetLab Host	Overall	CBT 80	RTT 80	CBT 60	RTT 60
pl001.ece.upatras.gr	0.199 015	0.215 197	0.216 825	0.221 980	0.240 031
plab4.ple.silweb.pl	0.184 446	0.195 942	0.204 779	0.201 948	0.226 570
planetlab01.dis.unina.it	0.259 907	0.269 230	0.284 222	0.278 992	0.314 478
planetlab-tea.ait.ie	0.187 084	0.200 495	0.203 818	0.206 327	0.230 746
planetlab1.cs.vu.nl	0.187 583	0.200 415	0.197 935	0.208 524	0.226 052
planetlab1.diku.dk	0.284 464	0.297 602	0.302 527	0.309 440	0.328 614
planetlab1.ifi.uio.no	0.186 199	0.199 308	0.199 447	0.200 740	0.230 393
planetlab1.jcp-consult.net	0.233 693	0.243 166	0.260 692	0.258 907	0.284 457
planetlab1.mta.ac.il	0.182 876	0.198 539	0.202 894	0.208 898	0.231 756
planetlab1.nrl.eecs.qmul.ac.uk	0.286 637	0.300 413	0.312 971	0.308 138	0.341 799
planetlab2.cs.uit.no	0.188 270	0.198 852	0.207 177	0.202 327	0.230 516
planetlab2.csg.uzh.ch	0.209 478	0.217 312	0.225 333	0.228 513	0.251 160
planetlab2.lkn.ei.tum.de	0.181 973	0.196 858	0.201 980	0.204 572	0.225 884
planetlab2.virtues.fi	0.189 057	0.196 824	0.205 922	0.204 714	0.228 961
planetlabpc0.upf.edu	0.190 232	0.196 679	0.207 544	0.206 092	0.227 695
ple01.fc.univie.ac.at	0.185 660	0.196 517	0.211 548	0.205 981	0.228 111

Table 6.5: Gini coefficients for CBT and Circuit-RTT methods on various measurement hosts.

PlanetLab Host	Overall	CBT 80	RTT 80	CBT 60	RTT 60
pl001.ece.upatras.gr	0.837 078	0.657 093	0.679 943	0.511 467	0.484 869
plab4.ple.silweb.pl	0.829 875	0.647 222	0.675 470	0.503 969	0.479 915
planetlab01.dis.unina.it	0.862 057	0.673 483	0.715 146	0.527 773	0.506 617
planetlab-tea.ait.ie	0.830 044	0.649 576	0.676 604	0.501 597	0.482 042
planetlab1.cs.vu.nl	0.830 060	0.644 623	0.677 833	0.496 262	0.481 240
planetlab1.diku.dk	0.871 530	0.680 434	0.712 514	0.528 831	0.505 199
planetlab1.ifi.uio.no	0.829 933	0.647 782	0.674 687	0.504 659	0.479 974
planetlab1.jcp-consult.net	0.853 047	0.673 306	0.694 492	0.519 609	0.493 390
planetlab1.mta.ac.il	0.829 548	0.655 753	0.678 187	0.499 161	0.483 876
planetlab1.nrl.eecs.qmul.ac.uk	0.872 292	0.684 033	0.713 484	0.522 840	0.506 293
planetlab2.cs.uit.no	0.829 755	0.650 527	0.673 490	0.505 571	0.476 520
planetlab2.csg.uzh.ch	0.839 976	0.654 464	0.684 837	0.503 159	0.486 667
planetlab2.lkn.ei.tum.de	0.829 647	0.644 824	0.671 730	0.498 465	0.477 575
planetlab2.virtues.fi	0.829 929	0.647 987	0.672 887	0.506 151	0.476 406
planetlabpc0.upf.edu	0.830 362	0.652 544	0.675 619	0.504 662	0.480 623
ple01.fc.univie.ac.at	0.829 602	0.643 441	0.673 290	0.497 954	0.480 917

Table 6.6: Normalized Shannon entropies for CBT and Circuit-RTT methods on various measurement hosts.

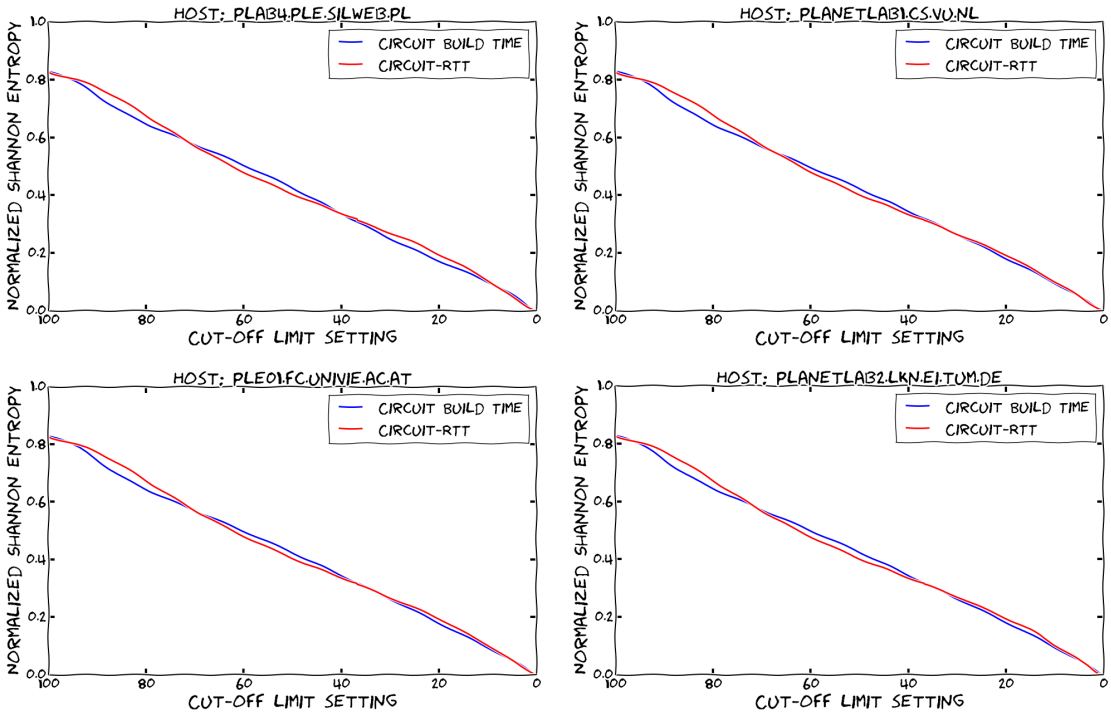

Figure 6.12: Normalized entropies for all possible CBT and Circuit-RTT cut-off limit settings.

cut-off limit at all, which is a major benefit, as such hard cut-off limits have detrimental effects on anonymity.

Figure 6.13 on the next page depicts the Gini coefficients for different CBT and Circuit-RTT cut-off limits. With lower cut-off limits, the difference in the Gini coefficient between CBT and Circuit-RTT increases slightly, in favor of CBT. For this reason, we conclude that Circuit-RTT biases slightly more towards certain nodes than CBT.

Comparing Circuit-RTT to CBT, we note that there is a minor skew in selecting certain nodes, but an improved selection probability of entry/exit-combinations with the default cut-off limit.

6.8 Guard Nodes

The last question we tackle regards to the use of Guard nodes. We aim to answer whether the selection of Guard nodes has a significant effect on latency as experienced by users. We used the data gathered from previous experiments, described in Section 6.2 on page 49, and grouped the observed nodes. Before analyzing the difference between those groups, we restricted our evaluation to Guard nodes for which we know at least 1000 TTFB values.

We could not use the widely used Analysis of variance (ANOVA) method, as it assumes data to fit a normal distribution, which in this case it does not. Instead, we use the Kruskal-Wallis

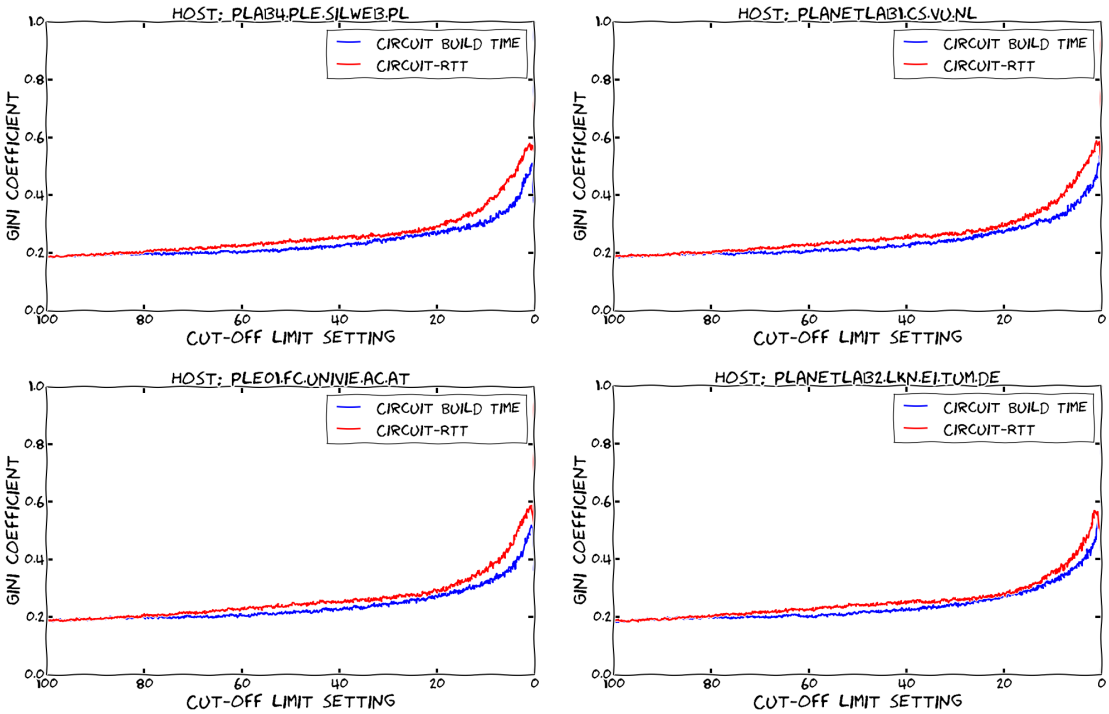

Figure 6.13: Gini coefficient for different CBT and Circuit-RTT cut-off limit settings.

test, a nonparametric test that compares groups by ranking their values first. This test does not rely on any specific distribution. The Kruskal-Wallis test does assume the groups' values are identically shaped, which holds in our case.

Results

Table 6.7 on the next page lists the Kruskal-Wallis rank sum and the number of Guards examined for each host.[11] As we have at most 260 groups, the maximal critical value of the χ^2 distribution is 493 [12]. We note that the Kruskal-Wallis rank sum is significantly above that value for every host, and so we conclude that the selection of Guard nodes has an influence on the latency as experienced by users. For every measurement host we get a p-value $< 2.2e^{-16}$, representing the chance that random sampling would result in a sum of ranks at least as big as the observed. Hence, we can conclude that the TTFBs for Guard nodes have different distributions.

[11] See footnote 7 on page 53 .

[12] Using a probability of 0.999 999 999 999 999 9.

PlanetLab Host	Kruskal-Wallist Rank Sum	# of Guards
pl001.ece.upatras.gr	77 581.25	237
plab4.ple.silweb.pl	87 715.67	260
planetlab01.dis.unina.it	27 622.9	115
planetlab-3.iscte.pt	1401.524	30
planetlab-tea.ait.ie	76 803.21	257
planetlab1.cs.vu.nl	83 938.25	257
planetlab1.diku.dk	17 880.96	91
planetlab1.ifi.uio.no	87 377.35	259
planetlab1.jcp-consult.net	43 416.04	163
planetlab1.mta.ac.il	81 636.02	254
planetlab1.nrl.eecs.qmul.ac.uk	19 234.07	93
planetlab2.cs.uit.no	83 500.98	257
planetlab2.csg.uzh.ch	65 211.01	210
planetlab2.lkn.ei.tum.de	90 251.38	257
planetlab2.s3.kth.se	85 575.51	257
planetlab2.virtues.fi	89 697.38	259
planetlabpc0.upf.edu	87 044.55	258
ple01.fc.univie.ac.at	92 539.31	258
ple2.ait.ac.th	60 041.75	255

Table 6.7: Kruskal-Wallis Rank Sum and number of Guards considered for all PlanetLab hosts.

CHAPTER 7

Discussion

> *"The time has come," the Walrus said,*
> *"To talk of many things:*
> *Of shoes — and ships — and sealing-wax —*
> *Of cabbages — and kings —*
> *And why the sea is boiling hot —*
> *And whether pigs have wings."*

Lewis Carroll, *Through the Looking-Glass, and What Alice Found There*, 1871

In this Chapter we discuss the limitations of our evaluation and provide an outlook on possible future work.

7.1 Limitations

Relay Adversary

Before any modification can be integrated into Tor, it is essential to study its effect on the anonymity the system provides. Any additional possibility for attackers to influence the path selection of clients has to be studied, and a reasonable trade-off between performance and anonymity has to be found. A potential disadvantage of the Circuit-RTT approach is that it is susceptible to manipulations by nodes, as a malicious exit node can identify the measurement probes and use this knowledge to influence results. The CBT approach already allows any malicious nodes to identify circuit creation cells easily, and to artificially delay the forwarding of

cells, so that the node appears congested, thus biasing the clients circuit selection. When Circuit-RTT is used, only the exit node can identify the measurement cells so easily. Other nodes would have to assume that the first cell relayed is the measurement.

Simulation

We assume that every client would benefit from using the Circuit-RTT method. However, since we conducted our experiments on the live Tor network using at most 19 concurrent clients, we cannot predict the changes the network would undergo if every client were to use that method. For this reason, whole-network emulation with ExperimenTor [69] or simulation with Shadow [70], which we consider future work, would be required to come to any conclusion.

Latency Attack

As noted in Section 4.2 on page 26, previous research found that algorithms that improve throughput or responsiveness of circuits also increase the effectiveness of latency-based attacks. We assume that the latency improvement from the implementation of Circuit-RTT will also increase the effectiveness of latency-based attacks, but we did not evaluate this possibility further.

Alternative Anonymity Metrics

To quantify anonymity, we used the classic entropy model, as described in Section 5.5 on page 46. However, an adversary controlling multiple ASes or IXPs, such as corporations, intelligence agencies, or countries, can have a dramatic effect on users' anonymity. Recent anonymity metrics thus model adversaries that control a fixed number of resources, such as nodes, ASes, IXPs, or a combination thereof, over a certain time period. Probabilistic analysis is then used to quantify the resulting level of anonymity. However, according to recent research results, using a congestion-aware circuit selection algorithm similar to Circuit-RTT only makes a minor difference in the time elapsed until the first circuit is successfully compromised. Nevertheless, future work should consider security metrics with regard to such network adversaries and should present probabilities of compromise over time. [cf. 39]

Guard Nodes

Basically Guard nodes were only considered in our evaluation in such sense that we aimed to use the same path selection as Tor would, as presented in Section 5.2 on page 38. No evaluation on how Circuit-RTT works with different sets of Guard nodes respectively a single Guard node has been conducted. However, single Guard nodes might be the default setting in Tor in the future, as pointed out in Section 3.1 on page 15 and Section 4.4 on page 34. Hence, it might be interesting to see how Circuit-RTT and also CBT perform for both sets of three and single Guard nodes.

Accuracy of Approximation

We have shown that also RTT values can be approximated by a Fréchet distribution, like CBT values. For this reason, we assume that the same approximation of the Fréchet distribution's tail by a Pareto distribution can be applied. However, as Pareto can only be used to approximate the tail of a Fréchet distribution, its application is limited to that area. Nevertheless, it remains to be shown that our assumption that the 50th to 80th percentile of RTT values can be accurately approximated with a Pareto distribution.

7.2 Future Work

Unreachable Destinations

As specified in the nodes' exit policies, exit nodes usually only allow connections to certain pairs of IP addresses and TCP ports. Although we consider this to be unlikely, there may exist some pairs that become unreachable due to the use of the Circuit-RTT method. However, to the best of our knowledge, no such evaluation has been conducted for CBT either, and so we consider it to be future work.

Improving Quality of Service

Bandwidth Limit Detection When conducting the experiment involving numerous RTT measurements on individual circuits, we encountered spikes at 100 ms and 1000 ms intervals on some circuits. Since these values correspond to the token bucket refill intervals, as described in Section 3.3 on page 20, these spikes might indicate that one or more nodes in the circuit are being asked to forward more traffic than they can handle. For this reason, we believe that RTT measurements could also be used to optimize circuit selection towards high bandwidth circuits. As we focus on improving latency in general in this thesis however, we consider such bandwidth limit detection to be future work.

Latency In addition to the bandwidth limit detection, multiple RTT measurements could furthermore improve latency for applications requiring a certain quality of service, such as VoIP applications. Such extensive measurements might prove able to increase the probability of certain latency properties, although not generally practicable due to the increased load on the network.

Number of Measurements

We have shown that the distribution of observed RTTs can be approximated by a GEV distribution. However, we did not come to a conclusion whether a maximum number of measurements should be taken or what value would be most suitable. Instead we stuck to a maximum of 1000, currently also used for CBT in Tor, since this seemed reasonable and increased the comparability of our results. Furthermore, we also did not examine the minimum number of measurements

required to reasonably estimate the parameters of the distribution. The knowledge of this minimum could lead to faster use of the method by clients after start-up.[1]

Guard Nodes

As we found the method of Guard node selection to have a significant effect on latency as experienced by users, involving RTT measurements in the Guard node selection algorithm could improve latency overall. However, in that we reduce the number of potential Guard nodes and increase vulnerability to the predecessor attack in connecting to a greater number of entry nodes, anonymity would be weakened. On the other hand, from a network topology point of view, since a Guard selection algorithm using RTT measurements would probably result in a bias towards nodes closer to the clients, some network-based attacks might be weakened. This alone might justify such an approach.

Circuit Cells

An alternative approach could be used to measure the RTTs of circuits. This approach would use only the final pair of circuit construction cells, rather than measuring the time to establish a complete circuit, as in CBT, or sending an additional cell after circuit completion, as in Circuit-RTT. This would measure the circuit's RTT only once, biasing only towards CPU-congested exit nodes, and would not require sending an additional cell for measurement. This would, however, be limited by a single measurement value per circuit.

Stream Attachment Failures

We did not explore reasons for TCP stream attachment failures as we did for circuit build failures, whose cause we were able to identify as inter-relay connectivity issues (see Section 8.1 on the next page). We did, however, encounter such failures as we were conducting the experiment involving multiple RTT measurements on individual circuits. Those failures seem eventually to affect the majority of circuits at least once. We could not find a relation to the nodes' stable flags, but undertook no detailed evaluation whatsoever, considering it future work.

AS and Country Traversal

In general, both CBT and Circuit-RTT comprise a trade-off between performance and anonymity. We assume that besides avoiding slower nodes respectively circuits also the traversal of ASes and countries is changed. To the best of our knowledge no such evaluation has been conducted yet.

[1] The minimum number of measurements for CBT in Tor currently is 100.

CHAPTER 8

Conclusion

> *And where does the newborn go from here? The net is vast and infinite.*

Major Motoko Kusanagi/Puppet Master, *Ghost in the Shell (by Kazunori Itō)*, 1995

One of the most important goals of Tor's design is to provide a low latency and high throughput transport service that supports interactive applications for activities, such as web browsing, which comprises up the vast majority of connections in the Tor network. However, occasional high end-to-end latencies not only are harmful for those delay-sensitive interactive web users, but also prevent the use of real-time applications like VoIP altogether, for which a certain quality of service is indispensable. As part of our study, we conducted several experiments on the live Tor network and analyzed the influence of the CBT and the Circuit-RTT methods on the quality of circuits with regard to latency, bandwidth, and anonymity.

8.1 Contributions

By making minor modifications to Tor's source code, we built a path generator that creates paths as an unmodified Tor client would without actually building a circuit. This path generator is an important part of the software we wrote to measure the latency and bandwidth achievable through Tor circuits. It could also be used, for example, to validate that other path simulators, such as TorPS [65], are selecting paths as Tor would. One of our modifications to Tor's source

code, which allows a client to clear its list of currently used Guard nodes, has proven to be of general use and thus has been merged into Tor version 0.2.5.1-alpha.[1]

Our measurement software makes heavy use of parallelization, in order to reduce the time required to probe numerous circuits. As a result of our effort to push the Tor software and the Tor control protocol to their limits, it became feasible to probe millions of circuits within a few days.[2]

Performance

The basic idea of the Circuit-RTT approach is that clients create a local view of achievable latency by actively measuring the circuits' RTTs. After a circuit has been established, the client measures the circuit's RTT and discards that circuit if its RTT is above a certain cut-off value. This ensures that a user gets a fast, preemptively built circuit when it is needed.

To reduce the bias introduced by a single, central destination server, we used Google's web servers and CloudFlare's CDN for measuring the latency and the bandwidth of circuits. Especially the CloudFlare's CDN could also prove useful for Tor's bandwidth scanners, since currently only a single server is used.

With the use of the Circuit-RTT method at the default cut-off limit we have observed improvements in every single metric: latency, bandwidth, and anonymity! Decreasing the cut-off limit can improve latency and bandwidth even further, but with some adverse effects on anonymity.

In order to carry out our measurements, we deployed several hosts on PlanetLab, allowing us to infer that the improvements we measured hold independently of the client's location.

Multiple Measurements

Although latency, bandwidth, and anonymity can be improved using the Circuit-RTT method, a single RTT measurement only provides a hint about the real quality of a circuit. More effort is required to ensure high quality. But such efforts, such as the use of multiple measurements, are in general unfeasible, as they would increase the load on the Tor network disproportionally if every client used them. Such in-depth measurements could only be justified for applications such as VoIP, requiring real-time service.

Guard Nodes

We have shown that the selection of Guard nodes, which was introduced to mitigate both the efficiency of the predecessor attack and the threat of statistical profiling, has a significant effect on latency as experienced by users.

[1] This functionality might prove to be useful for other software, as well, such as Tor metrics, which needs to select paths as Tor would, while leaving Guard nodes disabled.

[2] In another experiment that is not part of this thesis, we pushed the parallelization capabilities of our circuit performance measurement software to its limits: Examining the inter-relay connectivity between all Tor nodes, we measured tens of millions of circuits within three weeks, using only one host computer running a single Tor client instance in the process. [see 71]

8.2 Suggested Implementations

In order to incorporate our findings, we suggest two changes to the current implementation. Both can be deployed incrementally, so that any client can immediately use them without requiring a network-wide upgrade. Furthermore, the two changes are not dependent on one another, so that they can be implemented separately and independent of each other.

Replace CBT method by Circuit-RTT

Since the queuing and processing delays within many nodes fluctuate dramatically over time, it is vital to use a client-centric method that can react fast to changing conditions, such as the CBT or the Circuit-RTT method. The use of the Circuit-RTT method alone had better results than the CBT and a combination of CBT and Circuit-RTT. We thus suggest completely replacing CBT with Circuit-RTT.

With the default cut-off limit of 80%, the use of the Circuit-RTT method achieved improvements in latency, bandwidth, and anonymity. Compared to using the CBT method, using Circuit-RTT improved the average latency by 5.6% from 333ms to 315ms and the 90th percentile by 8.3% from 528ms to 484ms. Average bandwidth improved by 2.7% from 2.15Mbit/s to 2.21Mbit/s, and the 90th percentile improved by 3.6% from 1.12Mbit/s to 1.16Mbit/s. If we leave aside a negligibly small bias towards certain nodes, anonymity improved by 4.0% from 0.65 to 0.68 in terms of normalized Shannon entropy.

As they only require that a single cell be sent from a client to the exit node and back, RTT measurements are very lightweight. Hence, the additional load added to the Tor network can be assumed to be very small. Furthermore, it should be possible to reuse the complete code in Tor for the statistical calculations on CBT for Circuit-RTT without requiring any modifications - only the input values would change from CBTs to RTTs. From a practical point of view this can be considered a major benefit, as it should significantly reduce the work required on the implementation.

Configurable Cut-Off Limit

Currently, the cut-off limit setting can only be altered at compile-time. Making that option configurable[3] before execution or during runtime would offer different performance options for applications. This would enable users to select the appropriate performance option for their applications, whether that be web browsers, Internet chats, or even anonymous VoIP applications.

On the one hand, a range of configuration options may attract users with different needs. On the other hand, additional distinction between users would make partitioning attacks more feasible. For this reason, we suggest that the configuration option allow values within a restricted range from 50% to 80%. Reducing the cut-off limit further would certainly lead to an excessive increase in CPU resources used on nodes, as clients would create more circuits overall. Furthermore, reducing the cut-off limit considerably below 50% would require a new approach for the statistical approximations. Nevertheless, that increase in resource usage might still be

[3]For example as argument to the *"SOCKSPort"* configuration option, similar to the *"isolation flags"* argument.

considered justified if lower latency and higher throughput is desirable from the point of view of network design.

When the cut-off limit was reduced to 60%, average latency improved by 16.7% from 315ms to 262ms, and the 90th percentile improved by 16.5% from 484ms to 404ms, compared with the use of the Circuit-RTT method using a cut-off limit setting of 80%. Average bandwidth improved by 17.5% from 2.21Mbit/s to 2.67Mbit/s, and the 90th percentile improved by 14.5% from 1.16Mbit/s to 1.36Mbit/s. We observed a noticeable bias of 10% towards certain nodes and an anonymity decrease by 29% from 0.68 to 0.49, in terms of normalized Shannon entropy.

In conclusion, we argue that keeping the performance costs associated with the anonymity system as low as possible would make Tor more attractive both to existing users, who would use Tor more regularly, and to potential users who might join the network. Because the degree of anonymity provided by such system is closely tied to the number of its users, expanding the user base would ultimately enhance the system's anonymity properties.

Bibliography

[1] Vinton G. Cerf and Robert E. Khan. "A Protocol for Packet Network Intercommunication". In: *IEEE TRANSACTIONS ON COMMUNICATIONS* 22 (1974), pp. 637–648.

[2] Y. Rekhter et al. *Address Allocation for Private Internets*. RFC 1918 (Best Current Practice). Updated by RFC 6761. Internet Engineering Task Force, Feb. 1996. URL: `http://www.ietf.org/rfc/rfc1918.txt`.

[3] David Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". In: *Communications of the ACM* 24.2 (Feb. 1981).

[4] Ceki Gülcü and Gene Tsudik. "Mixing E-mail With Babel". In: *Proceedings of the Network and Distributed Security Symposium - NDSS '96*. IEEE, Feb. 1996, pp. 2–16.

[5] George Danezis, Roger Dingledine, and Nick Mathewson. "Mixminion: Design of a Type III Anonymous Remailer Protocol". In: *Proceedings of the 2003 IEEE Symposium on Security and Privacy*. May 2003, pp. 2–15.

[6] Ulf Möller et al. *Mixmaster Protocol — Version 2*. IETF Internet Draft. July 2003.

[7] George Danezis. "Statistical Disclosure Attacks: Traffic Confirmation in Open Environments". In: *Proceedings of Security and Privacy in the Age of Uncertainty, (SEC2003)*. Ed. by Gritzalis et al. IFIP TC11. Athens: Kluwer, May 2003, pp. 421–426.

[8] Dogan Kesdogan, Dakshi Agrawal, and Stefan Penz. "Limits of Anonymity in Open Environments". In: *Proceedings of Information Hiding Workshop (IH 2002)*. Ed. by Fabien Petitcolas. Springer-Verlag, LNCS 2578, Oct. 2002.

[9] Michael Reiter and Aviel Rubin. "Crowds: Anonymity for Web Transactions". In: *ACM Transactions on Information and System Security* 1.1 (June 1998).

[10] Philippe Boucher, Adam Shostack, and Ian Goldberg. *Freedom Systems 2.0 Architecture*. White Paper. Zero Knowledge Systems, Inc., Dec. 18, 2000.

[11] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: The Second-Generation Onion Router". In: *Proceedings of the 13th USENIX Security Symposium*. Aug. 2004.

[12] Andrei Serjantov and Peter Sewell. "Passive Attack Analysis for Connection-Based Anonymity Systems". In: *Proceedings of ESORICS 2003*. Oct. 2003.

[13] Andreas Pfitzmann and Marit Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf. Aug. 2010.

[14] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. "Hiding Routing Information". In: *Proceedings of Information Hiding: First International Workshop*. Ed. by R. Anderson. Springer-Verlag, LNCS 1174, May 1996, pp. 137–150.

[15] The Tor Project, Inc. *Tor Metrics Portal: Users*. https://metrics.torproject.org/users.html. [Online; accessed 13-Mar-2014]. 2014.

[16] Kevin Bauer. "Improving Security and Performance in Low Latency Anonymity Networks". PhD thesis. University of Colorado, May 2011.

[17] Roger Dingledine and Steven J Murdoch. *Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it*. https://www.torproject.org/press/presskit/2009-03-11-performance.pdf. [Online; accessed 02-June-2013]. 2009.

[18] Prithula Dhungel et al. "Waiting for Anonymity: Understanding Delays in the Tor Overlay". In: *Peer-to-Peer Computing*. 2010, pp. 1–4.

[19] Tao Wang et al. "Congestion-aware Path Selection for Tor". In: *Proceedings of Financial Cryptography and Data Security (FC'12)*. Feb. 2012.

[20] Damon McCoy et al. "Shining Light in Dark Places: Understanding the Tor Network". In: *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*. Ed. by Nikita Borisov and Ian Goldberg. Leuven, Belgium: Springer, July 2008, pp. 63–76.

[21] Andriy Panchenko and Johannes Renner. "Path Selection Metrics for Performance-Improved Onion Routing". In: *Proceedings of the 2009 Ninth Annual International Symposium on Applications and the Internet*. SAINT '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 114–120. ISBN: 978-0-7695-3700-9. DOI: 10.1109/SAINT.2009.26. URL: http://dx.doi.org/10.1109/SAINT.2009.26.

[22] Andriy Panchenko, Fabian Lanze, and Thomas Engel. "Improving Performance and Anonymity in the Tor Network". In: *Proceedings of the 31st IEEE International Performance Computing and Communications Conference (IPCCC 2012)*. Dec. 2012.

[23] Andrei Serjantov and George Danezis. "Towards an Information Theoretic Metric for Anonymity". In: *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Ed. by Roger Dingledine and Paul Syverson. Springer-Verlag, LNCS 2482, Apr. 2002.

[24] Claudia Diaz et al. "Towards measuring anonymity". In: *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Ed. by Roger Dingledine and Paul Syverson. Springer-Verlag, LNCS 2482, Apr. 2002.

[25] Corrado Gini. "Measurement of inequality of incomes". In: *The Economic Journal* 31.121 (1921), pp. 124–126.

[26] Robin Snader and Nikita Borisov. “A Tune-up for Tor: Improving Security and Performance in the Tor Network”. In: *Proceedings of the Network and Distributed Security Symposium - NDSS ’08*. Internet Society, Feb. 2008.

[27] Brent Chun et al. “Planetlab: An overlay testbed for broad-coverage services”. In: *ACM SIGCOMM Computer Communication Review* (2003), pp. 3–12.

[28] Roger Dingledine et al. *Tor: The Second-Generation Onion Router (2012 DRAFT*. `https://people.torproject.org/~karsten/tor-design-2012-11-14.pdf`. [Online; accessed 10-February-2014]. Nov. 2012.

[29] David Koblas and Michelle R Koblas. “Socks”. In: *UNIX Security III Symposium (1992 USENIX Security Symposium)*. 1992, pp. 77–83.

[30] The Tor Project, Inc. *Tor Metrics Portal: Network*. `https://metrics.torproject.org/network.html`. [Online; accessed 17-February-2014]. 2014.

[31] The Tor Project. *Tor directory protocol, version 3*. `https://gitweb.torproject.org/torspec.git/blob/HEAD:/dir-spec.txt`. [Online; accessed 26-February-2014]. 2014.

[32] Fallon Chen and Joseph Pasquale. “Toward improving path selection in Tor”. In: *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE. 2010, pp. 1–6.

[33] Stevens Le Blond et al. “One bad apple spoils the bunch: exploiting P2P applications to trace and profile Tor users”. In: *Proceedings of the 4th USENIX workshop on Large-scale exploits and emergent threats: botnets, spyware, worms, and more* (2011).

[34] Matthew Wright et al. “Defending Anonymous Communication Against Passive Logging Attacks”. In: *Proceedings of the 2003 IEEE Symposium on Security and Privacy*. May 2003, pp. 28–43.

[35] Lasse Øverlier and Paul Syverson. “Locating Hidden Servers”. In: *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS, May 2006.

[36] Matthew Wright et al. “The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems”. In: *ACM Transactions on Information and System Security (TISSEC)* 4.7 (Nov. 2004), pp. 489–522.

[37] Nicholas Hopper George Kadianakis. *The move to a single guard node*. `https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/236-single-guard-node.txt`. [Online; accessed 15-May-2014]. 2014.

[38] Tariq Elahi et al. “Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor”. In: *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2012)*. Raleigh, NC, USA: ACM, Oct. 2012.

[39] Aaron Johnson et al. “Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries”. In: *Proceedings of the 20th ACM conference on Computer and Communications Security (CCS 2013)*. Nov. 2013.

[40] Lasse Øverlier and Paul Syverson. “Improving Efficiency and Simplicity of Tor circuit establishment and hidden services”. In: *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*. Ed. by Nikita Borisov and Philippe Golle. Ottawa, Canada: Springer, June 2007.

[41] Kevin Bauer et al. “Low-Resource Routing Attacks Against Tor”. In: *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)*. Washington, DC, USA, Oct. 2007.

[42] Steven J. Murdoch and George Danezis. “Low-Cost Traffic Analysis of Tor”. In: *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.

[43] Roger Dingledine and Nick Mathewson. *Tor Path Specification*. `https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=path-spec.txt`. [Online; accessed 31-March-2014]. 2014.

[44] John Douceur. “The Sybil Attack”. In: *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS 2002)*. Mar. 2002.

[45] Karsten Loesing, Mike Perry, and Aaron Gibson. *Bandwidth Scanner specification*. `https://gitweb.torproject.org/torflow.git/blob/HEAD:/NetworkScanners/BwAuthority/README.spec.txt`. [Online; accessed 03-June-2013]. 2013.

[46] Christopher Wacek et al. “An Empirical Evaluation of Relay Selection in Tor”. In: *Proceedings of the Network and Distributed System Security Symposium - NDSS’13*. Internet Society, Feb. 2013.

[47] Jörg Lenhard, Karsten Loesing, and Guido Wirtz. “Performance Measurements of Tor Hidden Services in Low-Bandwidth Access Networks”. In: *Proceedings of the 7th International Conference on Applied Cryptography and Network Security (ACNS 09), Paris-Rocquencourt, France, June 2-5, 2009*. Ed. by Michel Abdalla et al. Vol. 5536. Lecture Notes in Computer Science. June 2009. ISBN: 978-3-642-01956-2.

[48] A. Panchenko, L. Pimenidis, and J. Renner. “Performance Analysis of Anonymous Communication Channels Provided by Tor”. In: *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*. Mar. 2008, pp. 221–228. DOI: `10.1109/ARES.2008.63`.

[49] Tao Wang and Ian Goldberg. “Improved Website Fingerprinting on Tor”. In: *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2013)*. Berlin, Germany: ACM, Nov. 2013.

[50] S. Cheshire and M. Krochmal. *Special-Use Domain Names*. RFC 6761 (Proposed Standard). Internet Engineering Task Force, Feb. 2013. URL: `http://www.ietf.org/rfc/rfc6761.txt`.

[51] M. Cotton and L. Vegoda. *Special Use IPv4 Addresses*. RFC 5735 (Best Current Practice). Obsoleted by RFC 6890, updated by RFC 6598. Internet Engineering Task Force, Jan. 2010. URL: `http://www.ietf.org/rfc/rfc5735.txt`.

[52] Mike Perry. *Use end to end stream time data to further prune circuits*. https://trac.torproject.org/projects/tor/ticket/5707. [Online; accessed 17-May-2013]. 2013.

[53] MaxMind. *GeoLite*. https://dev.maxmind.com/geoip/legacy/geolite. [Online; accessed 03-June-2013]. 2013.

[54] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. "LASTor: A Low-Latency AS-Aware Tor Client". In: *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. May 2012.

[55] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. "How Much Anonymity does Network Latency Leak?" In: *Proceedings of CCS 2007*. Oct. 2007.

[56] John Geddes, Rob Jansen, and Nicholas Hopper. "How Low Can You Go: Balancing Performance with Anonymity in Tor". In: *Proceedings of the 13th Privacy Enhancing Technologies Symposium (PETS 2013)*. July 2013.

[57] Robin Snader and Nikita Borisov. "Eigenspeed: Secure peer-to-peer bandwidth evaluation". In: *8th International Workshop on Peer-to-Peer Systems (IPTPS'09)*. 2009.

[58] Steven J. Murdoch and Piotr Zieliński. "Sampled Traffic Analysis by Internet-Exchange-Level Adversaries". In: *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*. Ed. by Nikita Borisov and Philippe Golle. Ottawa, Canada: Springer, June 2007.

[59] Matthew Edman and Paul F. Syverson. "AS-awareness in Tor path selection". In: *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009*. Ed. by Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis. Chicago, Illinois, USA: ACM, Nov. 2009, pp. 380–389. ISBN: 978-1-60558-894-0.

[60] Nick Feamster and Roger Dingledine. "Location Diversity in Anonymity Networks". In: *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2004)*. Washington, DC, USA, Oct. 2004.

[61] B. Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954 (Informational). Internet Engineering Task Force, Oct. 2004. URL: http://www.ietf.org/rfc/rfc3954.txt.

[62] Joshua Juen. "Protecting anonymity in the presence of autonomous system and internet exchange level adversaries". MA thesis. University of Illinois at Urbana-Champaign, Sept. 2012.

[63] Stefan Köpsell. "Low Latency Anonymous Communication – How Long Are Users Willing to Wait?" In: *Emerging Trends in Information and Communication Security*. Ed. by Günter Müller. Vol. 3995. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 221–237. DOI: 10.1007/11766155_16. URL: http://dx.doi.org/10.1007/11766155_16.

[64] The Tor Project. *TC: A Tor control protocol (Version 1)*. https://gitweb.torproject.org/torspec.git/blob/HEAD:/control-spec.txt. [Online; accessed 23-December-2013]. 2013.

[65] Aaron Johnson. *The Tor Path Simulator (TorPS)*. https://torps.github.io/. [Online; accessed 06-May-2014].

[66] Damian Johnson. *Stem*. https://stem.torproject.org/. [Online; accessed 30-May-2014].

[67] Susan J Devlin, Ramanathan Gnanadesikan, and Jon R Kettenring. "Robust estimation and outlier detection with correlation coefficients". In: *Biometrika* 62.3 (1975), pp. 531–545.

[68] The Tor Project, Inc. *Tor Metrics Portal: Performance*. https://metrics.torproject.org/performance.html. [Online; accessed 28-July-2014]. 2014.

[69] Kevin Bauer et al. "ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation". In: *Proceedings of the USENIX Workshop on Cyber Security Experimentation and Test (CSET 2011)*. Aug. 2011.

[70] Rob Jansen and Nicholas Hopper. "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation". In: *Proceedings of the Network and Distributed System Security Symposium - NDSS'12*. Internet Society, Feb. 2012.

[71] Roger Dingledine. *Measure connectivity patterns between relays*. https://trac.torproject.org/projects/tor/ticket/12131. [Online; accessed 04-June-2014]. 2014.

Acronyms

ANOVA Analysis of variance. 63

AS Autonomous System. 28–34, 68, 70

BGP Border Gateway Protocol. 29–31, 33

BSD Berkeley Software Distribution. 7, 15

CBT Circuit Build Time. 20, 36–38, 43, 44, 47, 49–54, 56–64, 67–71, 73

CC Creative Commons. iii

CDF Cumulative Distribution Function. 46

CDN Content Delivery Network. 5, 45, 59, 72

CIDR Classless Inter-Domain Routing. 18

COGS Changing of the Guards. 39

CPU Central Processing Unit. 25, 59, 70, 73

DNS Domain Name System. 12

FTP File Transfer Protocol. 9

GEV Generalized Extreme Value. 49–52, 54, 55, 69

GPL GNU General Public License. iii

GUI Graphical User Interface. 21

HTML HyperText Markup Language. 5, 45, 59

HTTP Hypertext Transfer Protocol. 5, 13, 45, 46, 51

HTTPS Hypertext Transfer Protocol Secure. 13

I/O Input/Output. 38, 40

IP Internet Protocol. 1, 2, 4, 8–10, 12, 13, 16, 18, 23, 26, 30, 31, 33, 38, 41, 42, 44, 69, 82

ISP Internet Service Provider. 29, 30, 33, 34, 44, 50

IXP Internet eXchange Point. 28, 29, 31–34, 68

LINX London INternet Exchange. 31

LZO Lempel-Ziv-Oberhumer. 42

NAT Network Address Translation. 1

OP Onion Proxy. 8

OSI Open Systems Interconnection. 12

PCA Principal Component Analysis. 28

PDF Probability Density Function. 20, 49–52

RFC Request for Comments. 1, 23, 24

RTT Round-Trip-Time. 5, 23–26, 33, 36–38, 40–45, 47–64, 67–74

SOCKS Socket Secure. 8, 12, 13, 42–45

SSH Secure Shell. 3, 9

tar tape archive. 42

TBB Tor Browser Bundle. 13

TCP Transmission Control Protocol. 8, 10–13, 38, 41–45, 48, 69, 70

Tor The Onion Router. 3–5, 7–13, 15–33, 35, 36, 38–47, 49–53, 56, 60, 61, 67–74, 82

ToS Type of Service. 31

TTFB Time-To-First-Byte. 38, 40, 42–45, 50–59, 63, 64

VoIP Voice over IP. 4, 5, 49, 69, 71–73

WSP Weighted Shortest Path. 26, 30