An Improved Algorithm for Tor Circuit Scheduling

by

Can Tang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2010

Abstract

Tor is a popular anonymity-preserving network, consisting of routers run by volunteers all around the world. It protects Internet users' privacy by relaying their network traffic through a series of routers, thus concealing the linkage between the sender and the recipient. Despite the advantage of Tor's anonymizing capabilities, it also brings extra latency, which discourages more users from joining the network.

One of the factors that causes the latency lies in Tor's circuit scheduling algorithm, which allows busy circuits to crowd out bursty circuits. In this work, we propose and implement a more advanced scheduling algorithm which treats circuits differently, based on their recent activity. In this way, bursty circuits such as those used for web browsing can gain higher priority over busy ones such as used for bulk transfer; the performance for most activities over Tor is improved, while minimal overhead is incurred. Our algorithm has been incorporated into the latest build of Tor.

Acknowledgements

I would like to thank my supervisor, Dr. Ian Goldberg, for his great assistance and support in my research and grad study. He guided me through everything from research ideas to theoretic fundamentals and coding techniques. I feel very lucky to have such a great supervisor.

I would like to thank Ryan Henry, Femi Olumofin, Aniket Kate and Greg Zaverucha for their comments on the research work and early drafts of this thesis. I would like thank The Tor Project for their financial support, and also for incorporating our results into the main Tor code. I would also like to thank my thesis readers Dr. Urs Hengartner and Dr. Srinivasan Keshav. I would like to acknowledge MITACS and NSERC for their financial support as well.

I thank my friends Shihong Yan, Penghao Diao, Kun She, Yu Wang, Run Yuan, Zhe Chen, Xin Gao, Feifei Ling, Bo Wang, Ran Zheng, Ruoyu Hou, Kexin Hou, Guanlong Li, Mengyuan Dai, He Chen, Wanying Luo, and Qi Xie, for making my life in Waterloo a very enjoyable experience.

Finally, I would like to thank my parents for supporting everything I do in my life, for giving me courage when I was depressed, for sacrificing a lot to give me the best education I could have when I was a child, and for caring about me like I was around while I was away from home. They don't ask me for anything, but are always there when I'm in need. For me, they are the best parents in the world.

Contents

List of Figures **xi**

1 Introduction **1**

2 Overview of Anonymity Networks and Tor **3**

2.1 Mix Networks . . . 3

2.2 Anonymous Remailers . . . 4

2.2.1 Type-0 Remailers . . . 4

2.2.2 Type-I Remailers . . . 5

2.2.3 Type-II Remailers . . . 5

2.2.4 Type-III Remailers . . . 5

2.3 Low-latency Anonymity Networks . . . 6

2.3.1 Anonymizer . . . 6

2.3.2 Jondonym/AN.ON . . . 6

2.3.3 Onion Routing . . . 6

2.4 Tor . . . 7

2.5 Adversary Model and Attacks . . . 8

2.6 Tor's Performance Issues . . . 9

2.7 Summary . . . 10

3 Prioritizing Interactive Circuits **13**

3.1 Incentives . . . 13

3.2 How Tor's Circuits Work . . . 14

3.3 Circuit Selection based on EWMA . . . 15

4 Experiments and Results **17**

4.1 Implementation Notes 17

4.2 PlanetLab Simulations 18

4.2.1 End-to-End Timing Analysis 18

4.2.2 Fine-grained Timing Analysis 21

4.3 Experiments on Live Tor Nodes 24

4.3.1 Bandwidth Requirement for Live Tor Nodes 24

4.3.2 Testbed Setup 25

4.3.3 Experimental results 26

4.3.4 Effects on Bulk Transfer 26

4.4 Overhead 29

4.5 Compatibility with the Existing Tor 31

5 Fine-tuning of the Algorithm **33**

5.1 Testbed Setup 33

5.2 Experimental Results 34

6 Effects on Hidden Services **37**

6.1 Overview 37

6.2 Experimental Results 39

7 Related Work **41**

7.1 Tor over DTLS 41

7.2 Opportunistic Bandwidth Measurement 42

7.3 Performance-Based Path Selection 42

7.4 Providing Incentives for Tor Relays 43

7.5 Internet QoS Schemes 43

8 Security Issues **47**

8.1 Traffic Analysis 47

8.2 Strategy Proofness 48

9 Future Work **49**

9.1 Reduce the Cell's Time in the Output Buffer 49

9.2 Prioritize Connections 50

10 Summary **53**

References **53**

List of Figures

4.1 The Tor circuit structure for Experiment 1 19

4.2 CDF for time cost of downloading a small file for unprioritized and prioritized Tor 19

4.3 CDF for time cost of downloading a small file for unprioritized and prioritized Tor, under traffic simulation 20

4.4 Time points for a cell's life cycle in the middle OR, unprioritized 22

4.5 Time points for a cell's life cycle in the middle OR, prioritized 22

4.6 Cumulative incoming and outgoing cells of the circuit queue in the middle OR, unprioritized 23

4.7 Cumulative incoming and outgoing cells of the circuit queue in the middle OR, prioritized 23

4.8 Time cells wait in the circuit queue in the middle OR 24

4.9 CDF for time cost of downloading a small file using unprioritized and prioritized middle OR, in the real Tor network 26

4.10 CDF for time cost of downloading a small file using unprioritized and prioritized middle OR, in the real Tor network, during afternoons ET 27

4.11 CDF for time cost of downloading a small file using unprioritized and prioritized middle OR, in the real Tor network, during midnights ET 27

4.12 CDF for time cost of bulk download using unprioritized and prioritized middle OR, in the real Tor network 28

4.13 CDF for time cost of downloading a small file for unprioritized and prioritized Tor, under CPU-limited scenario 30

5.1 Comparison of performance for different values of H: CDF 34

5.2 Comparison of performance for different values of H: latency vs. H. Values for unprioritized Tor are shown by horizontal lines. 35

6.1 Overview of Hidden Services. Each arrow is a three-hop connection over the Tor network. 38

6.2 CDF for time cost of connection initialization and webpage fetching for Hidden Services, for unprioritized and prioritized Tor 39

Chapter 1

Introduction

Tor [DMS04] is a distributed anonymizing network that provides privacy for its users. The network is formed by volunteers all around the world running Onion Routers (ORs). The ORs publish their related information, such as bandwidth and exit policies, to a set of centralized servers called directory authorities. These directory authorities negotiate with each other and reach a consensus about the ORs. An end user runs an Onion Proxy (OP) locally to tunnel application requests through Tor; the OP downloads the consensus from the directory authorities, randomly picks a set of ORs, choosing them weighted by their reported bandwidths, and builds circuits through them. Then application traffic is relayed through the circuit, using layered encryption. In this way, a single OR only sees its previous OR/OP and next OR/OP, but not the actual sender-recipient relationship, and the privacy of the user at the transport level is preserved.

The anonymity provided by Tor relies on the size of the *anonymity set* — the set of users who could have performed a given action. Currently, there are around 1500 ORs [Tor09], and an estimated quarter million Tor users. Tor's rapid expansion period ended by the end of 2007 (in terms of the number of ORs) [Loe09b]. After that, Tor entered a relatively stable stage: the number of ORs joining the Tor network roughly equaled the number leaving it. One of the obstacles for Tor's further expansion is its performance issues.

There are many causes for Tor's performance issues. One of the causes is that bursty circuits do not co-exist with busy circuits very well. When multiple circuits are sharing a single connection between two ORs, busy circuits such as those for bulk transfer will greatly degrade the performance of the bursty ones, such as those for web browsing. Although this effect is inevitable due to the limited bandwidth resources, we want to allocate the resources more efficiently; that is, to give higher priority to the circuits with low throughput or short bursts of traffic, and make them faster. This is reasonable from the application point of view as well: circuits for web browsing or instant messaging are usually sensitive to delays, while those for bulk transfer are usually not.

Our approach is to calculate the exponentially weighted moving average (EWMA) for the number of cells sent in each circuit. When selecting the circuit to process, we always pick one with the lowest EWMA value, and flush the cells in that circuit. Newly created circuits and bursty circuits will usually have a low EWMA value, and so they will be prioritized.

Experiments showed that our improvement is effective in both PlanetLab simulation and the live Tor network. In the live experiment, the average time to download a small file decreased by 21% when we implement our enhancement on a single OR in a circuit.

We give an overview of anonymity networks in Chapter 2. In Chapter 3, we introduce the incentives and rationality of our proposal, briefly demonstrate the mechanism underlying Tor circuits, and propose our improvement. In Chapter 4, we show the results of some experiments under different scenarios, with analysis of the results and the overhead. Chapter 5 examines the effect of different parameter values on the performance of our system, and Chapter 6 explores the performance improvements on Tor's hidden services. Chapter 7 provides some related work on improving Tor's performance. Chapter 8 analyses the effects of our algorithm on Tor's security properties. We discuss possible future work in Chapter 9, and conclude in Chapter 10.

Chapter 2

Overview of Anonymity Networks and Tor

This chapter presents an overview of anonymity networks and Tor, to readers not familiar with them. A more thorough description of Tor can be found in [DMS04].

2.1 Mix Networks

David Chaum proposed the concept of mix networks [Cha81] to achieve anonymity for electronic mail. The proposed network consisted of multiple proxies called mixes. A message is multiply encrypted and sent through a path consisting of multiple mixes. At each mix, the message is decrpyted, padded to a uniform length, mixed into batches and sent out. Anyone watching the incoming and outgoing traffic on a mix, except the mix itself, will not be able to correlate the messages. In this design, unless all the mixes in the selected path collude, the message sender cannot be linked to the recipient.

Systems such as Babel [GT96], Mixmaster [MCPS03] and Mixminion [DDM03] adopted Chaum's mix network design. These systems provided protection against strong global adversaries. However, for interactive applications such as web browsing, remote login, or instant messaging, the delay introduced by reordering and mixing is unacceptable. In order to support anonymized interactive applications, the processes that greatly increase the overall delay should be removed from the design. Anonymizer [Ano10], Jondonym/AN.ON [JAP06], Onion Routing [GRS99] and Tor are examples of such a design, which provide low-latency services for interactive applications. These designs trade off the strong security properties for better usability.

Real-time response implies that an adversary watching both ends of the communication path can confirm the relationship between the sender and the recipient (traffic confirmation) [Dan03]; thus real-time anonymity networks only resist non-global adversaries. We will show later by examples of deployed anonymity networks, however, that such an adversary model is reasonable in practice, provided that the network is deployed at a large scale.

2.2 Anonymous Remailers

People's growing awareness of their privacy in the digital age can be exhibited in the following citation by Johan Helsingius, the operator of the Penet remailer (anon.penet.fi) [Gra94]:

> "Some people from a university network really argued about if everybody should put their proper name on the messages and everybody should be accountable, so you could actually verify that it is the person who is sending the messages. And I kept arguing that the Internet just doesn't work that way, and if somebody actually tries to enforce that, the Internet will always find a solution around it. And just to prove my point, I spent two days or something cooking up the first version of the server, just to prove a point."

As a result, starting from the early 90's, Chaum's initial design evolved into various anonymous remailer services, assisting people in preserving their anonymity in email messages. In this section, we take a brief look at the designs of these systems.

2.2.1 Type-0 Remailers

A *type-0 remailer* refers to a mail server that accepts email messages from clients, performs operations on the message to remove all the information that can identify the original sender, and relays the message to the final destination. In addition, the server can assign pseudonyms to the actual senders, and maintains a list of the sender-pseudonym relationships. Thus, the reply messages to the pseudonyms can be relayed back to the original senders. anon.penet.fi, located in Finland, was one of the emerging type-0 remailers in the early 90's. By September 1994, anon.penet.fi had over 700,000 registered users.

This type of remailer has the advantage of easy implementation, deployment and maintenance. However, the relayed plaintext message made it easy for eavesdroppers to read the content. Additionally, powerful adversaries can break into the server or exert legal pressure to the server operator to get the list revealing the linkage between the pseudonyms and the actual senders. In 1995 and 1996, the Church of Scientology claimed to be harmed by anonymous email sent from anon.penet.fi, and used legal pressure to force the operater to reveal the identity of the sender. After these two compromises, the operator felt unable to provide privacy for their users, and shut down the service.

2.2.2 Type-I Remailers

The next generation of remailers, type-I remailers, greatly improved the security compared to their predecessors. They used encryption for the messages to defeat casual eavesdroppers, removed message logs and identity information from the senders, and adopted chaining in the sending process: messages are relayed through a *chain* of remailers to the final destination, to avoid single point of attack.

This design is secure even against more powerful attackers: even large cooperations and government agencies cannot compromise the chain of remailers easily, since the remailers are distributed geographically.

There are still attacks available to compromise the anonymity of type-I remailers: the *size correlation* attack, which records the sizes of incoming messages and outgoing messages on a remailer, and correlates these messages using the sizes, and replay attack, which records the plaintext message, sends it to the remailer multiple times and watches which outgoing message repeated, so as to discover the corresponding encrypted message.

2.2.3 Type-II Remailers

In order to solve the vulnerabilities of type-I remailers, type-II remailers enhanced the design by dividing the messages into fixed-sized packets to defeat the size correlation attack, and by recording message IDs to prevent the replay attack. Mixmaster [MCPS03] is a representative type-II remailer service.

A big disadvantage of type-II remailers is their poor support for reply messages. The sender still need to use the insecure type-I style remailer to receive replies.

2.2.4 Type-III Remailers

Type-III remailers, represented by Mixminion [DDM03], have been proposed to address the problems of type-II remailers. The design further enhanced the security of type-II remailers. Since the recorded ID list in a type-II remailer needs to be kept short, old message IDs will expire after a certain amount of time, and the attacker can replay a message after its ID expires. In type-III remailers, key rotation is used to make the replay attack infeasible: the remailer will rotate its encryption key regularly, and record every message ID addressed to a given key before the key expires. Forward anonymity is ensured by using ephemeral keys; exit policies can be configured to prevent exit abuse. In addition, *single-use reply blocks* are adopted to support a secure way of receiving replies to messages sent anonymously.

2.3 Low-latency Anonymity Networks

Privacy protection for email, however, is not sufficient; message delays in the remailer network can be many hours, or even days. This is inappropriate for protecting other types of network traffic, such as web browsing, which require *low-latency* approaches to anonymity.

2.3.1 Anonymizer

The Anonymizer is the simplest design of a low-latency anonymity network. It consists of a single trusted server, acting as an anonymizing proxy. The server accepts messages from the user, strips identifying information, and sends it to the indended recipient. The reply messages are relayed in the reverse order. The design is similar to that of anon.penet.fi: it is simple and efficient, yet bears the same vulnerabilities as anon.penet.fi. Users need to trust the anonymizing server to keep their identities secret.

2.3.2 Jondonym/AN.ON

The Jondonym service is a branch of the AN.ON project [JAP06]. The Jondonym network consists of multiple mix servers. The mix servers are operated by independent organizations, and their identities are public. Among these mix servers, the user explicitly selects those she trusts and forms a *cascade*. The traffic from the user is encrypted for every mix server in the cascade and transferred through the cascade. A large number of users will share the same cascade, thus hiding them among each other.

One of the big advantages of Jondonym, compared to other anonymity networks, is that only certified persons and organizations can operate its mix servers. This to some extent prevents malicious mixes from joining the network, and provids some assurance of the qualities of mixes. On the other hand, this discourages more non-malicious mixes from joining the network as well, thus slowing down the speed of expansion.

2.3.3 Onion Routing

Onion Routing was proposed in 1996 [GRS99] to provide anonymity to interactive applications on the Internet, such as web browsing and telnet. The initiator of traffic randomly selects a set of geographically distributed Onion Routers (ORs) to form a path. Different from Jondonym, these ORs are run by volunteers who do not need to publish their identities, or get certified by a centralized server. Then initiator chooses a *path* of ORs, and creates an *onion* that contains an encrypted layer for each OR on the path, including the secret keys for each OR. Internet traffic is multiply encrypted and relayed through this path. Each OR decrypts one layer of the message and forwards the message to the next OR in the path. Return traffic is also relayed through the path, but in the reverse order, and the message is encrypted one more layer at each OR. The design evolved into the Tor network, which we will discuss in more detail in the next section.

2.4 Tor

The Tor network [DMS04] is the second generation Onion Routing network, and the most widely used anonymity network. The network consists of over 1000 relays (ORs) located around the world. The relays provide cryptographic operations on incoming messages and send them out without mixing and padding, to achieve low latency.

A client runs an Onion Proxy (OP) locally, to accept user-level TCP requests and relay data to the network. Before the OP can relay data, a *circuit* consisting of multiple ORs needs to be created. The circuit will be used to transport data chunked into *cells* of fixed length (512 bytes). The set of ORs in the circuit is chosen randomly, weighted by the reported bandwidth of each OR. Once the set of ORs is decided, the circuit is created incrementally: the OP builds a TLS connection to the first OR, and negotiates a symmetric key with it, by sending command cells containing the related cryptographic data. Once the symmetric key is negotiated, the OP extends the circuit to the second OR by telling the first OR to build a TLS connection to the second OR, and relay the command cells sent between the OP and the second OR. Thus a symmetric key with the second OR can be negotiated, without the second OR knowing the OP's IP address. In the same way, the circuit can be extended to the next OR, until the last OR (the exit OR) in the set is reached. The circuit creation phase is completed at this step.

After circuit creation, the OP can accept user-level requests. It breaks the data from the application layer into cells, encrypts the cells multiple times using the symmetric keys negotiated with each OR in the circuit, and sends the cell to the first OR. When receiving the cell, each OR decrypts the cell using its symmetric key, and forwards the cell to the next OR in the circuit. At the exit OR, the cell is unencrypted after decryption. The exit

OR regroups the cells into TCP streams, and forwards them to the destination server. The server only sees a TCP connection from the exit OR, without knowing the encryption and decryption processes in the circuit.

When receiving data from the server, the process is reversed: the exit OR accepts data from the server, breaks the data into cells, encrypts them and forwards them to the previous OR in the circuit. Each OR encrypts an extra layer when receiving the cell, and forwards the cell to its previous OR. The OP will receive the multiply encrypted cell, and decrypts it with the session keys to get the unencrypted cell. Then the cells will be regrouped into TCP streams and sent back to the application layer.

None of the ORs except the first (entry) OR knows the initiator of the circuit, and none of the ORs except the exit OR knows the destination of the traffic it relays. Without the presense of a global adversary, anonymity for the clients is achieved. But since the exit OR is the one who talks directly to the server, it is responsible for the behaviour of the actual client. In order to avoid abuse issues, the operator of an OR can set its exit policy, which specifies the type of traffic for which it is willing to serve as an exit OR.

Tor also has mechanisms for fairness, rate limiting and congestion control. ORs use a token bucket approach to ensure a long-term average bandwidth, at the same time permitting short bursts of traffic. Besides TCP's own congestion control mechanism, Tor provides both circuit-level throttling and stream-level throttling to achieve better utilization of network resources. These mechanisms alone, however, do not achieve the optimal utilization of network resources, since they ignore the fact that different Tor users have different bandwidth requirements and delay tolerances.

2.5 Adversary Model and Attacks

Since the data is grouped into fixed-size cells and encrypted at each OR, the appearance of a cell changes completely when going through an OR. Thus, an adversary cannot correlate cells based on their bit patterns. But since ORs do not mix cells or provide cover traffic, an adversary watching both ends of the circuit can correlate cells based on their timing pattern (end-to-end timing correlation).

In practice, in order for end-to-end timing correlation to work, the attacker needs to compromize tens of globally distributed ORs to have a non-negligible probability to compromize the anonymity of a certain Tor client, according to statistics [Tor09]. This attack is highly impractical even for powerful organizations such as government agencies.

For the same reason, it is difficult to launch DoS attacks, since the targets are distributed globally and form a large number. Alternatively, the adversary can choose to become part of the network himself, by volunteering as one or more ORs. If the adversary's bandwidth accounts for certain fraction of the whole network, he will have control over a corresponding portion of circuits. This is not easy to achieve either, since individual OR's bandwidth is capped to a threshold. The adversary needs to fund multiple ORs with a total bandwidth high enough to attract bulk traffic.

Using techniques described in [MD05], a modest adversary can still launch an attack against Tor to greatly compromise its anonymity. Instead of passively observing the traffic pattern, the adversary actively sends his own traffic through a Tor node. Since different circuits of traffic through a node interfere with each other, he can observe the change in latency in the target circuit in order to infer whether the node is in the circuit. The attack cannot directly discover the identity of the client, but will reduce the anonymity of Tor to a series of known proxies. In practice, as long as the client frequently switches to new circuits, this attack only has minimal success rate.

The security of the data content at the exit OR should be a significant consideration by Tor clients, since the data is decrypted into plaintext at the exit OR. A hostile exit OR can read the sensitive content such as username/password, and may modify the data. The end-to-end security should be ensured by using end-to-end secure transport protocols like TLS by the client.

To conclude, although Tor does not provide perfect anonymity, it provides enough protection in most practical scenarios. In fact, it is currently the most successful deployed anonymity network.

2.6 Tor's Performance Issues

Tor entered a rapid expansion stage since its inception in 2004: the number of relays increased linearly to around 1500 by the end of 2007, and the total bandwidth increased to around 400 MB/s [Loe09a]. But after that, Tor's fast pace of expansion ceased, and Tor has entered a relatively stable stage, in terms of number of ORs and total bandwidth. As of April 2010, the number of ORs was around 1400 and the total bandwidth was around 420 MB/s [Tor09].

Many reasons account for this cease of expansion, both political and technical. For example, the data retention laws passed in the beginning of 2008 in Germany discouraged many OR operators from continuing running the ORs, because of the uncertainty of whether running an OR remained legal. In 2009, the Chinese government blocked not only the Tor website, but the Tor network as well. The unavailability of the Tor website prevented Internet users in China from making themselves acquainted with Tor and downloading the Tor software, while blocking the Tor network made it technically very difficult to access Tor or volunteer as an OR in mainland China. As a result, this act almost eliminated all ORs in China.

Aside from political reasons, the main obstacle that keeps potential users and potential OR operators from joining Tor is its performance issues. Tor users experience much higher latency compared to those without Tor. The reason is multifold: the multi-hop relay structure of Tor's circuit certainly contributes to the overall latency, and this structure cannot be circumvented; besides, Dingledine and Murdoch [DM09] identified several reasons for Tor's bad performance. Some major ones are listed below:

- The Quality of ORs

  From [Tor09], we see that most of the network bandwidth comes from only a small portion of ORs. For the majority of low-bandwidth ORs, they are likely to have low quality in performance, yet considering their number, they still have chances to be selected by some Tor clients. Such ORs will certainly degrade the performance of a circuit once selected.

- Usage Abuse

  Tor is intended to provide an anonymizing service for low latency, interactive applications, but still a large portion of bandwidth is used for bulk transfer. [MBG+08] This adds a large amount of traffic to the network, slowing it down for everyone.

- Capacity Limitation

  Currently, the Tor network does not have enough capacity to satisfy all the user requests: the OR/user ratio is very low. Although it is made very easy for a client to become an OR, there is no direct incentive for the user to donate her bandwidth to the Tor network.

- Path Selection

  Tor's path selection algorithm is not optimal under real network situations. The selection is based on each OR's reported bandwidth, which may not be accurate. Using only the bandwidth as the criterion, not others such as latency and geolocation, may not result in the optimal path in terms of performance either.

- Network Overhead

  Clients need to download and update regularly the directory information from directory servers, which contains information about each OR. This overhead may still be high for low-bandwidth users.

- Extra Overhead for Hidden Services

  Different from regular Tor services, accessing Hidden Services (see Chapter 6) involves an extra layer of indirection, which adds to the performance overhead.

2.7 Summary

With more and more people realizing the need to protect their online privacy, privacy enhancing technologies have developed rapidly in the past two decades. Based on Chaum's model, several anonymity networks were designed and deployed to provide anonymity for Internet users. These networks are divided into two categories: high-latency anonymity networks and low-latency (real-time) anonymity networks.

Tor is the most successfully deployed low-latency anonymity network. It provides anonymity on the IP level for TCP-based applications. Currently there are over 1000 ORs in the network, operated by volunteers.

There are serious performance issues in the Tor network, which slows down Tor's further expansion. Our work will concentrate on improving the performance of Tor, and providing a better user experience.

Chapter 3

Prioritizing Interactive Circuits

In this chapter, we propose our improvement on the circuit scheduling algorithm. First, we state the incentives for our proposal; next, we describe the process by which Tor ORs select which data is to be transmitted. Finally, we describe our improvement over the existing system.

3.1 Incentives

Most users of Tor experience its performance issues: it incurs much higher latency than direct connections. Although the multi-hop architecture inevitably brings extra latency, the experienced latency is higher than this effect can explain.

One factor for Tor's bad performance is its limited capacity: the ORs are run by volunteers, usually on consumer computers, with limited bandwidth. Within this limited capacity, there are abuse issues observed in the Tor network. According to McCoy et al. [MBG$^+$08], a small number of BitTorrent connections consume a very high proportion of Tor bandwidth. These connections make Tor unusable for many potential users.

Tor provides anonymity by mixing a specific user into a crowd of users; therefore, the degree of anonymity Tor provides depends on the number of users. Higher latency will discourage more users from joining the network. Hence, the performance issues do not only affect user experience, but also degrade Tor's security properties.

The uses of Tor can be divided into interactive streams and non-interactive streams. Interactive streams include web browsing, instant messaging, SSH, and telnet, while non-interactive streams include bulk file transfer such as FTP and BitTorrent. Interactive streams are usually delay-sensitive: users click on a link to a webpage and wait, expecting it to appear on the screen in seconds, while non-interactive streams are not: BitTorrent users expect the file download to be completed in hours or even days; they can tolerate higher delays. We aim to improve Tor's performance by making ORs process interactive streams first. This will give interactive users (the majority of Tor users) a better experience, and will make little difference for non-interactive users.

3.2 How Tor's Circuits Work

As described in Chapter 2, a client runs an OP locally. The OP randomly selects several ORs to form a path, then builds a Tor circuit through this path. Each circuit is used by only one client (OP). Between each pair of ORs on the path, a Tor connection is established. If multiple circuits use the same two ORs in sequence, they will share a single connection between the two ORs. Based on the current number of users and ORs, and the ORs' capacities, we can infer that usually connections will be shared by multiple circuits, especially for the connections between high-bandwidth ORs. That is, each OR will have simultaneous connections to a number of other ORs (but only one connection to any *given* OR), and each connection will transport data for a number of circuits.

All Tor traffic is relayed in fixed-size (512-byte) cells. A cell consists of a header field and a payload. The header contains metadata about the cell, such as the circuit identifier. When a cell arrives at an OR, the OR decrypts the cell, extracts the information necessary from the header, and then pushes the cell into the output queue for its circuit (the *circuit queue*). The time cost of this process is negligible [Rea08] as a fraction of the overall time for a cell to be processed by an OR. Circuits with non-empty circuit queues are called *active* circuits.

Each connection has an *output buffer*; data written to that buffer will be transmitted to the next OR in FIFO order. As multiple circuits generally share a single connection, the cells in the circuit queues must be multiplexed into the output buffer.

When there is room in the output buffer, the OR will select an active circuit, and move some cells from its circuit queue to the output buffer. If all of the cells are moved, the circuit is marked inactive.

The contribution of this work is to change how Tor decides from which active circuit to select cells. The previous algorithm was simply to select active circuits in round-robin fashion. We show that by making a more judicious selection, the performance of Tor for interactive circuits can be notably improved, while minimally affecting performance for circuits performing bulk data transfer, which tend to be delay insensitive in any event.

In order to prioritize interactive circuits, we need to decide, for example, which circuits are using HTTP, and which circuits are using BitTorrent. Unfortunately, we cannot determine the application protocol by looking directly at the content of the cells, since all of the cells are encrypted except for at the exit OR. On the other hand, circuits using HTTP may also perform bulk transfer, and we want to deprioritize them as well. Thus, the amount of traffic sent recently should be an appropriate criterion on which to base our scheduling decision. We should mention that we do not want to block BitTorrent or similar applications by blocking the port number at exit ORs, since Tor is intended to be application neutral; additionally, port number blocking can be easily circumvented by those file sharing applications.

3.3 Circuit Selection based on EWMA

We want to have a metric for "how many cells a circuit has sent recently", and base the circuit selection decision on this metric. The metric needs to represent an average value over a period of time of the activity for a circuit, and also needs to decay over time, since we do not want the activity from long ago to have a large impact on the current decision for a circuit. The exponentially weighted moving average (EWMA) seems to be a good choice for this metric.

First, we assign each circuit a cell count value, representing the average number of cells sent recently. Every time we wish to flush some cells to the connection's output buffer, we calculate the decayed cell count value for each circuit, based on the EWMA equation that supports irregularly-spaced observations:

$$A_{t+\Delta t} = A_t \cdot 0.5^{\frac{\Delta t}{H}}$$

where $A_{t+\Delta t}$ is the new cell count value, A_t is the old cell count value, Δt is the elapsed time since the last observation and H is the "half life" parameter; that is, H determines the interval after which the previous average is reduced by half.

After the calculation, the OR picks the circuit with the smallest cell count value, and flushes that circuit's cell queue to the output buffer of the connection; the cell count value is updated correspondingly:

$$A'_{t+\Delta t} = A_{t+\Delta t} + C_{t,t+\Delta t}$$

where $C_{t,t+\Delta t}$ is the number of cells sent in the interval $(t, t + \Delta t]$.

In our application, Δt is the interval between circuit queue flushes, and typically has a value in the range of milliseconds. The value of H, as well as a switch for turning our whole algorithm on or off, can be set in the Tor configuration file.

As the equation shows, circuits with low or bursty traffic will have low cell counts. These circuits are likely to be those which are still in their creation phase, as well as circuits for web browsing or instant messaging, which are exactly the circuits we want to prioritize. For each OR, cells in interactive circuits will wait for less time in the circuit queue than without prioritization. The performance of interactive circuits, on the whole, will be improved. As we show later, the performance of circuits which are deprioritized suffers only minimally.

In regards to the selection of the algorithm, we are not claiming that EWMA is the only right choice. We selected EWMA for its ease of implementation and effectiveness in practice, and show later in this work that it indeed performs well. Other options may also be appropriate, however, and we leave as future work the question of finding the optimal choice.

Chapter 4

Experiments and Results

4.1 Implementation Notes

We implemented the algorithm based on version 0.2.1.19 of Tor. We modified the function that selects a circuit among the set of active circuits in a connection and flushes cells from this circuit to the output buffer. The set of active circuits was stored as a circularly linked list. The simplified algorithm is described as Algorithm 1:

Algorithm 1 $flush\_from\_first\_active\_circuit(connection\ conn)$

```
$now \Leftarrow gettime()$
$circ \Leftarrow conn.active\_circuits$
$iter \Leftarrow circ$
$min\_ewma \Leftarrow \infty$
**repeat**
    $\Delta t \Leftarrow now - iter.last\_time$
    $iter.ewma \Leftarrow iter.ewma \times power(0.5, \Delta t/H)$
    $iter.last\_time \Leftarrow now$
    **if** $iter.ewma < min\_ewma$ **then**
        $min\_ewma \Leftarrow iter.ewma$
        $circ\_min\_ewma \Leftarrow iter$
    **end if**
    $iter \Leftarrow iter.next$
**until** $iter = circ$
$num\_flushed \Leftarrow flush\_circuit(circ\_min\_ewma)$
$circ\_min\_ewma.ewma \Leftarrow circ\_min\_ewma + num\_flushed$
**return** $num\_flushed$
```

The algorithm gets the current system time, and updates the EWMA count for each circuit. It then picks the circuit with the least EWMA count, flushes this circuit to the output buffer, and adds the number of flushed cells to its EWMA count accordingly. In the description, *iter* is the iterator going through the whole circuit list; *last_time* is the last time the circuit ewma count was updated; H is the parameter used in the EWMA algorithm we discussed in the previous chapter, which is retrieved from Tor's configuration file.

The size of the patch is 12 KB, which adds or modifies fewer than 200 lines of C code. The implementation does not rely on any libraries, thus not bringing extra overhead to deployment.

4.2 PlanetLab Simulations

The first part of the experiments were performed on PlanetLab. PlanetLab [PACR03] is a research network consisting of nodes distributed globally, much like Tor. We selected a set of five nodes from PlanetLab, and ran a private Tor network on them. A typical Tor circuit consists of three ORs. The OR directly connecting to an OP, the OR directly connecting to the server, and the OR in the middle are called the entry OR, exit OR, and middle OR, respectively. In our experiment, we picked two nodes as directory servers and three nodes as ORs; we ran our modified Tor on the three ORs.

4.2.1 End-to-End Timing Analysis

Experiment 1 was designed to measure the time cost of downloading a small file (simulating web browsing), while there are competing bulk transfer circuits. According to [Web08], the average web page size (including HTML, CSS and images) grew from 93KB to over 312KB, from 2003 to 2008. We picked 300KB as the file size we use for our experiment. The file is hosted on the same machine as the exit OR, in order to eliminate the variance introduced by the connection between the web server and the exit OR.

We configured three local clients, who select the same path, as shown in Figure 4.1. Two clients were performing bulk transfer. We tried to download the 300KB file using the other client, and recorded the elapsed time. We used $H = 66$ for the algorithm, which means that after every 66 seconds, the old cell count will decay by a factor of 0.5. This corresponds to our initial estimate that a decay of 10% over 10 seconds would be appropriate. We performed 100 downloads for both unprioritized Tor (the stock Tor) and our prioritized version of Tor. The cumulative distribution function (CDF) of the results is shown in Figure 4.2.

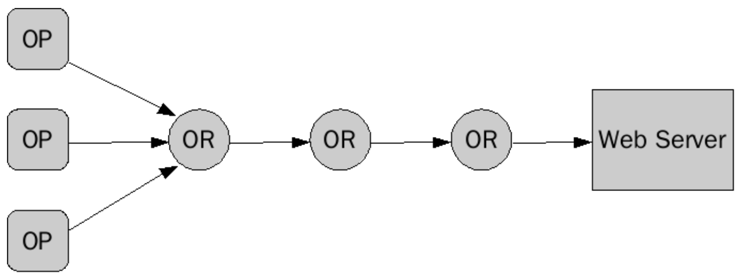

Figure 4.1: The Tor circuit structure for Experiment 1

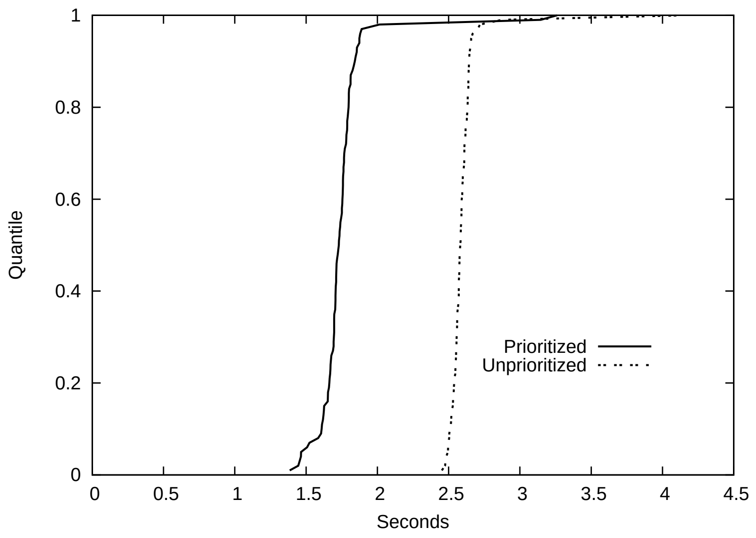

Figure 4.2: CDF for time cost of downloading a small file for unprioritized and prioritized Tor

In Experiment 1, we observed an average of 32% decrease in the time to download a small file while there are simultaneous competing bulk transfers — the median time decreased from 2.60 seconds to 1.75 seconds.

During the experiment, when switching between unprioritized and prioritized Tor, we switched on the algorithm for all three ORs at the same time. But does prioritizing one of the ORs contribute the most to the effect, or do all three of them contribute equally to the improvement? In order to investigate this issue, we selectively turned on the algorithm on individual ORs, and repeated the experiment to see the effect. We discovered that switching on the algorithm on the entry and exit ORs does not have noticeable effects, while by turning on the algorithm on the middle OR, we immediately see an improvement that is close to the overall improvement we obtained. The reason is that for the nodes we chose on PlanetLab, the link between the middle OR and the entry OR is slower than the

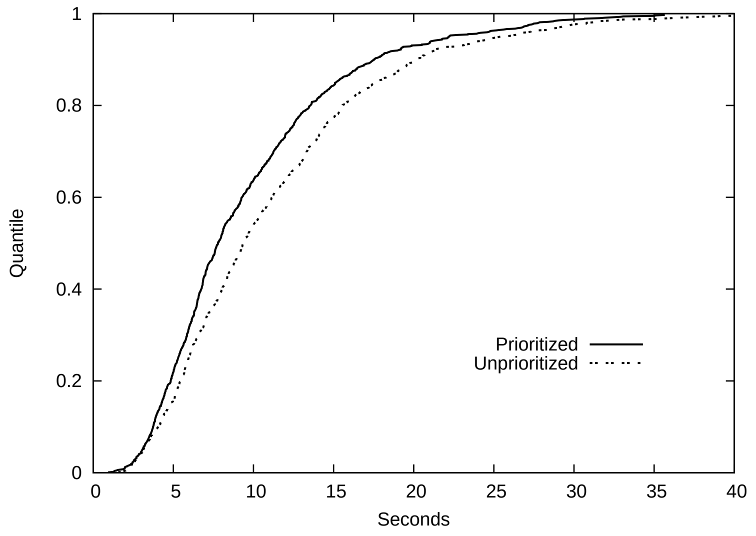

Figure 4.3: CDF for time cost of downloading a small file for unprioritized and prioritized Tor, under traffic simulation

link between the middle OR and the exit OR, so that cells (being sent from the server to the client) accumulated at the middle OR. This indicates that the effectiveness of the algorithm highly depends on the conditions of individual ORs; when an OR's output link is congested, cells will wait in the circuit queue for longer periods, and our algorithm will make a more noticeable difference.

For the real Tor network, the traffic distribution is quite different from the previous experiment. McCoy et al. [MBG+08] identified the exit traffic protocol distribution of Tor in 2008. HTTP accounted for 92.45% of all connections and 57.97% of the total bytes sent; SSL accounted for 4.06% of all connections and 1.55% of the total bytes sent; BitTorrent accounted for 3.33% of all connections and 40.20% of the total bytes sent. Other protocols such as Instant Messaging, E-Mail, FTP, and Telnet accounted for less than 1% of both connections and bytes sent. From the cited results, we can conclude that BitTorrent consumed a disproportionately large amount of bandwidth compared to other protocols. This justifies our intention of giving interactive streams higher priority. On the other hand, in reality, the ratio of busy circuits is not as high as in our previous experiment; i.e., there are not as many low-priority circuits. To see how much improvement we can get, we created a traffic simulator that randomly generates network traffic according to the connection-to-throughput ratio in the above statistics.

In Experiment 2, we ran the traffic simulator on multiple clients, to simulate the Tor network. We downloaded a small file and recorded the time cost. There were 1000 attempts for both unprioritized and prioritized Tor. The results are shown in Figure 4.3.

Due to the large variance in the network conditions, the results have large variance in both unprioritized and prioritized Tor (ranging from 1 second to almost 1 minute). However, from the CDF graph, we can still find 10–20% improvement at most quantiles.

4.2.2 Fine-grained Timing Analysis

In Experiment 3, we examine the life cycle of a cell in an OR, check how much time it spends at each stage, and see where we have improved.

Remember that when a cell reaches an OR, it enters the circuit queue to which it belongs, and waits to be flushed to the output buffer of the connection. When the output buffer is empty or has flushed some cells, one circuit will be selected to flush its cells to the output buffer. Then the cells wait in the output buffer until they are flushed to the socket.

The testbed setting is similar to Experiment 1. We use libspe [Rea09] to record the time points related to the cells: when cells enter the circuit queue, when they are moved from the circuit queue to the output buffer of the connection, and when they leave the output buffer. We record those time points of cells in the middle OR (as we mentioned, using our algorithm on the middle OR showed the most improvement in our experiment), for both unprioritized and prioritized Tor.

Figures 4.4 and 4.5 show the results for a single download of a small file. The x-axis indicates the time a cell enters the circuit queue, and the y-axis indicates the time the cell enters the circuit queue (the straight diagonal line), the time the cell moved from the circuit queue to the output buffer, and the time the cell is flushed from the output buffer. Thus the gap between the two lower lines indicates the time a cell spends in a circuit queue, while the gap between the upper two lines indicates the time a cell spends in the output buffer, waiting to be written to the socket. We can see that a cell spends most of its life cycle in the circuit queue waiting to be flushed. The prioritized Tor greatly reduced this duration.

Figures 4.6 and 4.7 show the cumulative number of incoming and outgoing cells for the circuit queue. The horizontal gap between the two lines “In” and “Out” represents the time a cell spends in the circuit queue, while the vertical gap shows the length of the circuit queue. We can observe that both versions have similar incoming patterns, but the prioritized version has a more steep slope for outgoing cells, which indicates that the prioritized version flushes cells from the circuit queue more quickly.

Next, we recorded the time cells spent in the circuit queue. We see significant improvement in these durations; in fact, the average duration spent in the circuit queue decreased from 653 milliseconds to 115 milliseconds. The actual improvement in latency is even greater than the improvement in the average duration. Figure 4.8 shows the amount of

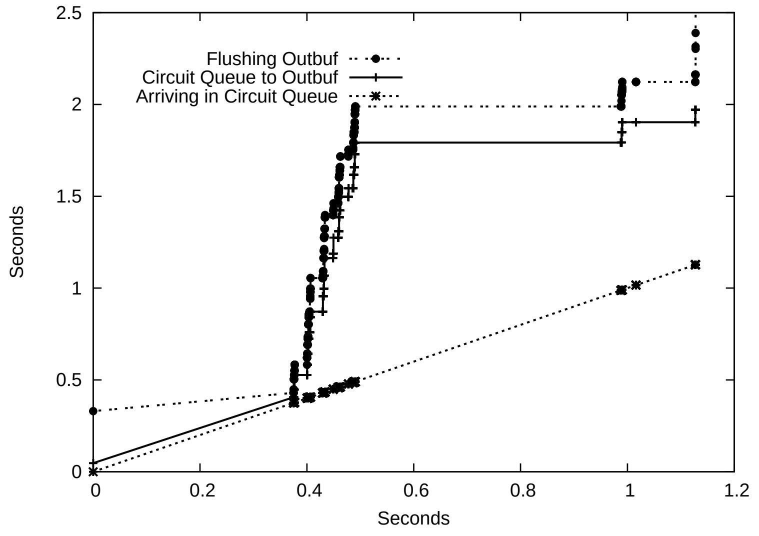

Figure 4.4: Time points for a cell's life cycle in the middle OR, unprioritized

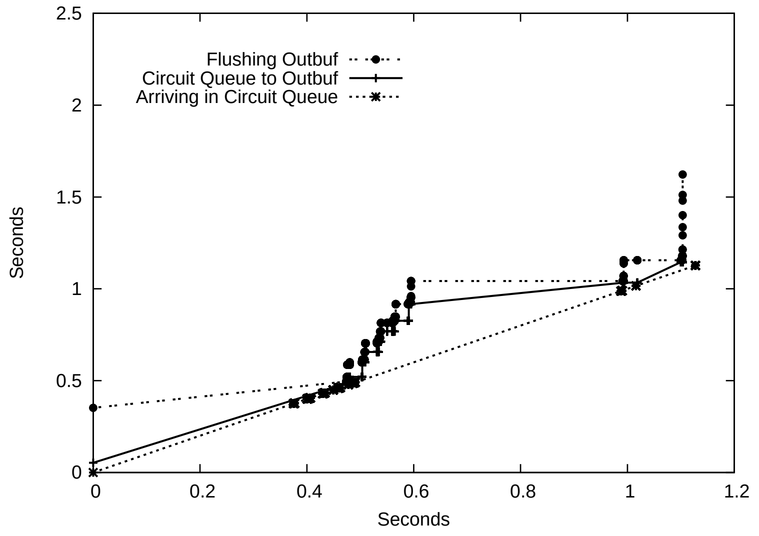

Figure 4.5: Time points for a cell's life cycle in the middle OR, prioritized

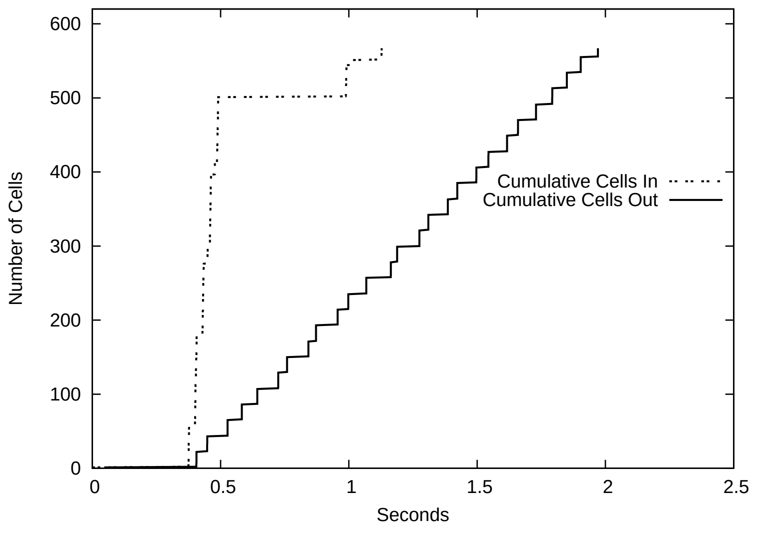

Figure 4.6: Cumulative incoming and outgoing cells of the circuit queue in the middle OR, unprioritized

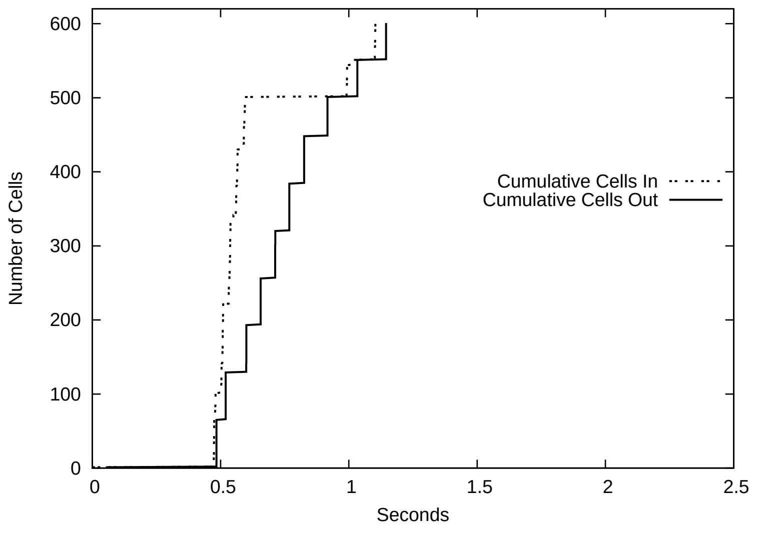

Figure 4.7: Cumulative incoming and outgoing cells of the circuit queue in the middle OR, prioritized

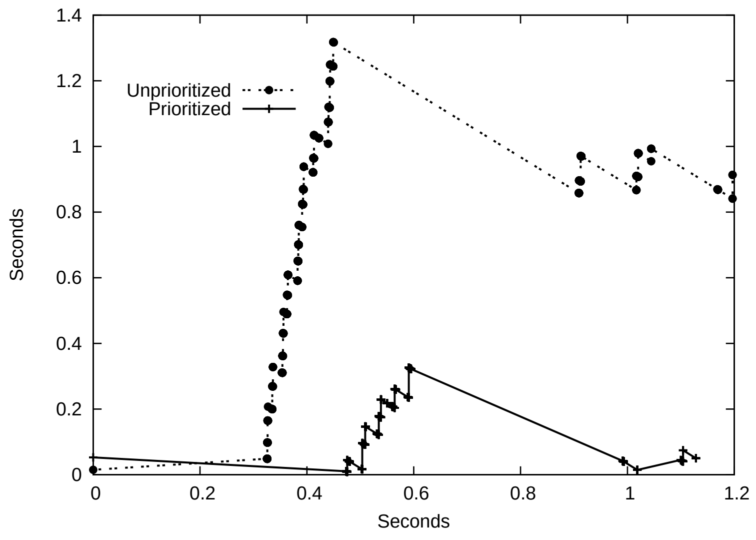

Figure 4.8: Time cells wait in the circuit queue in the middle OR

time cells wait in the circuit queue, which is just the difference between the lower two lines in each of Figures 4.4 and 4.5. We can see that as cells arrive, more cells are queuing up within the OR, making the latency higher. Our improvement is more noticeable when the cells are queuing up: the decrease in latency is as high as 1 second, in this case. This explains the results we obtained in Experiment 1.

4.3 Experiments on Live Tor Nodes

In order to test the effectiveness of our algorithm on the real Tor network, we need to perform the experiments on running Tor nodes, with real Tor traffic going through them. In this section, we describe our live experiments and results, and analyze the limitations.

4.3.1 Bandwidth Requirement for Live Tor Nodes

The effectiveness of our algorithm depends on whether multiple circuits are sharing a single connection between a pair of ORs. According to [Tor09], as of March 2010, there were around 1500 ORs in Tor network, with the highest reported bandwidth as much as 15 MB/s.

We downloaded the descriptors of all the ORs, and calculated the sum of the advertised bandwidth. The total bandwidth was 440 MB/s. We assume that there are 250,000 simultaneous Tor users (circuits), (this estimate comes from [Loe09b]) who select ORs randomly using their bandwidths as weights. Assume that we have a pair of ORs, each having bandwidth BW MB/s. The expected number of circuits between this pair of ORs is then $250000(\frac{BW}{440})^2 = 1.29(BW)^2$. In order to see several circuits between the pair to make the prioritization effective, we require $BW \approx 2$ MB/s at least.

The above calculation is only a rough estimate to ensure the ORs we select are in the right range. The accurate estimate of the number of circuits would be more complicated.

4.3.2 Testbed Setup

Initially, we planned to run three ORs on selected PlanetLab nodes, and let them join the Tor network. However, among the PlanetLab nodes, few could reach such a high bandwidth requirement. In fact, most nodes have bandwidth lower than 100 KB/s.[1] With these nodes, it is hardly possible to attract multiple circuits within the connection. We found one PlanetLab node at Princeton University that has bandwidth as high as 1 MB/s. However, the daily usage on the node is limited to 10 GB. Since the startup process of an OR requires several hours to complete (including publishing descriptors, computing consensus by directory authorities, advertising bandwidth and re-advertising bandwidth after attracting traffic), the throughput limit makes it impossible to perform any tests.

Instead, we used `gurgle.cs.uwaterloo.ca` (nickname: `planetgurgle`), a machine located at the University of Waterloo, which has a comparably high bandwidth of 3 MB/s. We also selected two existing ORs with high bandwidth: `blutmagie` with 10 MB/s, and `coldbotTorHosting1` with 10 MB/s. We used `blutmagie` as the entry OR, `planetgurgle` as the middle OR, and `coldbotTorHosting1` as the exit OR. In this way, the estimated number of circuits between `planetgurgle` and `coldbotTorHosting1` is $1.29(3 \times 10) \approx 39$, which is more than enough for our purpose.

The disadvantage compared to using PlanetLab nodes is that we could only control one OR in our circuit (the middle OR); therefore, the results will only show the improvement of prioritizing at one hop. However, as we will see later, the results are still noticeably in our favour.

The target file to fetch is hosted in the University of Waterloo, with a size of 87 KB. We did not introduce artificial bulk transfer traffic, so this experiment is representative of normal user experience. We used webfetch [Aiu04] to fetch the file using our configured circuit. There is a 20-second break between every successive fetch.

[1]This low bandwidth results from PlanetLab's bandwidth cap: since several slices share a single node, the bandwidth for each slice may be capped by the node's policy.

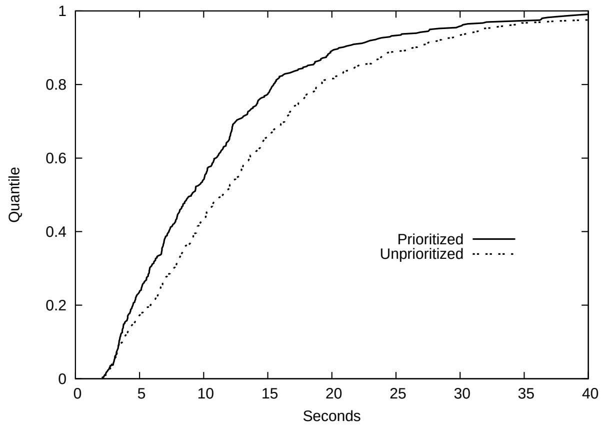

Figure 4.9: CDF for time cost of downloading a small file using unprioritized and prioritized middle OR, in the real Tor network

4.3.3 Experimental results

We performed the experiment during different periods in a day, executing 250 downloads with `planetgurgle` configured in each of the unprioritized and prioritized modes. The CDF of the results is shown in Figure 4.9.

It is interesting to note that this graph is quite similar to our traffic simulation tests on PlanetLab (Figure 4.3). With our prioritization algorithm enabled, the median time decreased from 11.49 seconds to 9.04 seconds. One phenomenon we observed is that during different periods in a day, the test results differ markedly. The latencies are much lower in the afternoons ET (Eastern Time; the timezone of New York and Toronto) than around midnight ET. This may indicate that most Tor users come from the other half of the globe. Indeed, according to [Loe09a], only a small percentage of Tor users come from North America and South America combined.

In order to better observe the effectiveness of our patch, we divided the results into two groups: a "fast" group which were performed during afternoons and a "slow" group which were performed around midnight. The CDFs are shown in Figure 4.10 and Figure 4.11.

The figures indicate that under various network conditions, our algorithm makes observable improvement for bursty HTTP downstream traffic.

4.3.4 Effects on Bulk Transfer

Our new scheduling algorithm should not degrade the performance of bulk transfer to any noticeable extent. By Little's Law [LG08], $L = \lambda W$, where L is the queue length (average

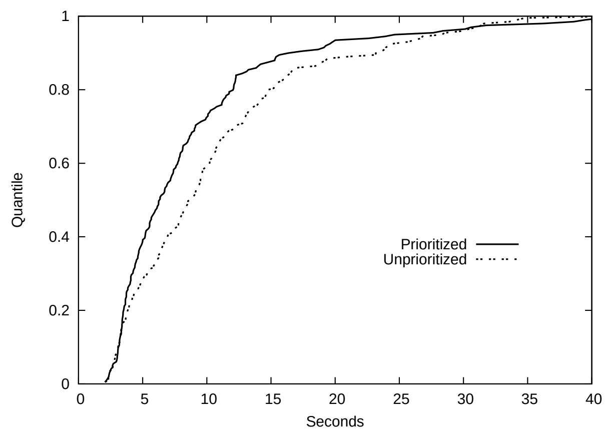

Figure 4.10: CDF for time cost of downloading a small file using unprioritized and prioritized middle OR, in the real Tor network, during afternoons ET

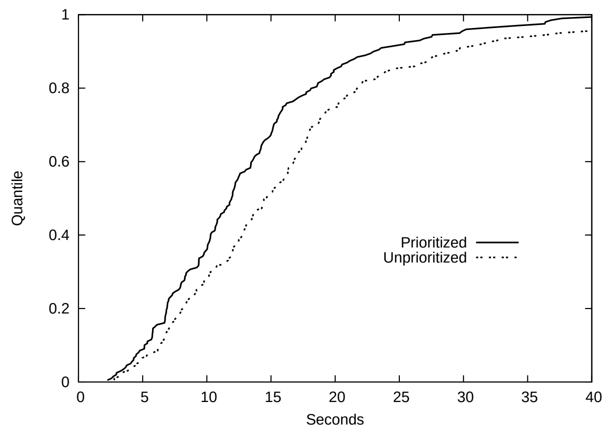

Figure 4.11: CDF for time cost of downloading a small file using unprioritized and prioritized middle OR, in the real Tor network, during midnights ET

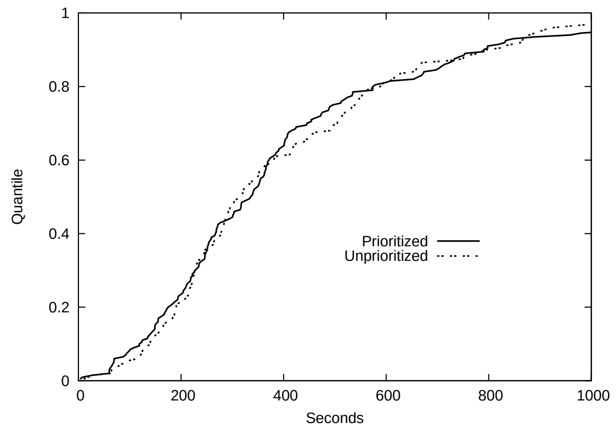

Figure 4.12: CDF for time cost of bulk download using unprioritized and prioritized middle OR, in the real Tor network

number of cells in the queue), λ is the arrival rate (long term throughput), and W is the average time a cell spends in the queue (latency). Our algorithm only changes the order of cells within a queue, thus does not change L and W. Under the assumption that the buffers are large enough, the long-term throughput for bulk transfer should stay the same.

We also experimentally compared the performance of bulk transfer circuits on our live node. We used our Tor client to continuously fetch a 4 MB file hosted at the University of Waterloo. There were 200 trials for each of unprioritized and prioritized Tor. The results are shown in Figure 4.12.

From the CDF, we see very little effect of our algorithm on bulk transfer. The average time cost is 416 seconds for unprioritized Tor, with standard deviation 335 seconds, and 419 seconds for prioritized Tor, with standard deviation 403 seconds. There is no statistically significant difference in the performance of unprioritized and prioritized Tor. In fact, the Kolmogorov-Smirnov (K-S) statistic [NIS10] for the two distributions is $0.065 < \sqrt{\frac{2}{N}}$, where $N = 200$ is the size of each sample. This indicates that the (two-sample) K-S test cannot confirm that the two samples are from different distributions. Additionally, as we mentioned earlier, bulk transfer usually takes at least several minutes to complete, and users doing such transfers will have more tolerance of the increased delay if they ever notice it at all.

4.4 Overhead

The overhead for our scheduling algorithm mainly lies in the computation of EWMA values, and the cost of acquiring the current system time. This requires extra CPU resources compared to the stock Tor. However, most Tor nodes are limited by the network capacity, not by their CPUs [Rea08]. Our scheduling algorithm will not degrade the performance of those nodes.

However, the Tor maintainers reported to us that the busiest Tor nodes are in fact CPU-limited. For these nodes, we need to make sure that prioritized Tor does not perform worse than the stock Tor. When we completed the first version of the scheduling algorithm, and performed local experiments (with very high network capacity, unlike the Tor network), we found that it did in fact perform worse than the stock Tor, and used a high ratio of CPU resources. We identified that the frequent calls to `gettimeofday` accounted for the majority of the time so consumed. When each cell is flushed, we need to know the system time in order to correctly update the EWMA values, but system calls at this frequency become a burden to the CPU.

We observed that during each write event (when cells are flushed into the output buffer), the differences in time of flushing for each of the cells in that write are usually in the handful of microseconds range. Since we do not need precision to the microsecond level for the calculation of EWMA values, we modified the algorithm so that we only acquire the system time at the beginning of the write event handling process, and store the time value. The subsequent acquisitions of system time use the cached value instead. In this manner, we reduced the total number of `gettimeofday` system call by two orders of magnitude.

After the optimization, we again performed a local experiment to find the overhead. The experiment was performed on a commodity desktop computer, with AMD Athlon 64 X2 Dual Core 5600+ processor, 3.2GB memory, Ubuntu 8.04 operating system. We ran all the Tor nodes, including three ORs, two directory authorities, and two OPs locally. The web server was also hosted locally. This setup maximally stresses the CPU. We performed the experiment in which the two clients simultaneously fetch a 5MB file from the web server. There were 200 trials for both unprioritized Tor and prioritized Tor. During the experiment, the CPU usage went up to 100%, so that our nodes were indeed CPU-limited. The CDF of the results is shown in Figure 4.13 ("Unprioritized" and "Prioritized (list)").

The results showed that the average time cost is 1.66 seconds for unprioritized Tor, with standard deviation 0.15 seconds, and 1.69 seconds for prioritized Tor, with standard deviation 0.24 seconds. There is no statistically significant difference in the performance of unprioritized and prioritized Tor in the local experiment, which means that even in the rare scenario that the Tor node is CPU-limited, the scheduling algorithm will not make it significantly slower.

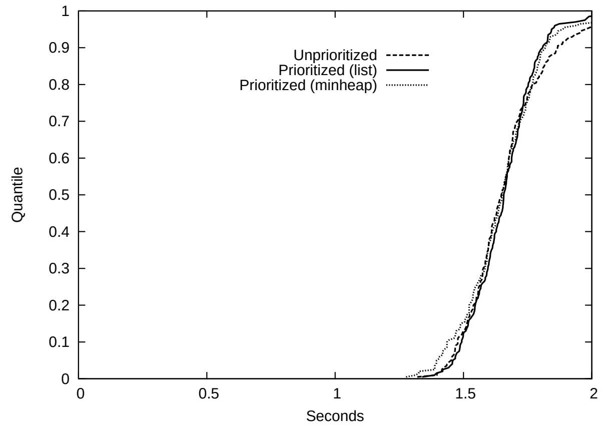

Figure 4.13: CDF for time cost of downloading a small file for unprioritized and prioritized Tor, under CPU-limited scenario

Nonetheless, Nick Mathewson of The Tor Project has optimized [Mat09] the implementation of our algorithm to further reduce the overhead. Instead of using a circular linked list, the active circuits are kept in a minheap-based priority queue. Further, the computation of the EWMA cell counts is optimized; noting that only the *relative*, and not the *absolute* EWMA cell counts matter (since their purpose is just to pick the active circuit with the lowest cell count), an arbitrary reference time point is picked, and cell counts are computed relative to that time. That is, rather than decaying every circuit's stored cell count value by a factor of $0.5^{\frac{\Delta t}{H}}$, and then adding 1 for every cell sent, a single new value V (representing the current weight of one cell, as compared to a cell sent at the reference time point) is updated by multiplying it by $0.5^{-\frac{\Delta t}{H}} = 2^{\frac{\Delta t}{H}}$. This value V is added to a circuit's cell count for each cell it sends. In our version, we traverse the list of active circuits and compute the decayed cell count for each one every time a circuit queue is flushed; in the optimized version, only the count for the circuit being flushed needs to be updated. Every so often, the cell counts for all circuits can be renormalized by dividing them all by V, and resetting V to 1. This has the effect of updating the reference time point to the current time. It is important to note that this optimized computation maintains exactly the same circuit-selection behaviour as our unoptimized implementation.

Mathewson's patch not only reduces the load of the EWMA computation, but also reduces the time cost of picking the highest-priority circuit when many circuits co-exist in a connection. We performed an experiment to test the overhead with this patch in the CPU-limited scenario. The results are also shown in Figure 4.13 ("Prioritized (minheap)"). The average time cost is 1.65 seconds, with standard deviation 0.16 seconds, also not a statistically significant difference. This version of our algorithm has been committed to the latest version (0.2.1.21) of Tor.

4.5 Compatibility with the Existing Tor

For any upgrade of a distributed system the size of Tor, compatibility is a fundamental issue to consider. Requiring simultaneous upgrades for all Tor nodes would be a great resisting force to the implementation. Fortunately, since our algorithm only changes the order that cells are multiplexed from different circuits within the OR, it does not require any change in other ORs. Consequently, each OR can be upgraded individually, and each upgrade would make some improvement, as can be seen by the results of our experiments.

Our algorithm can also be turned on and off conveniently, by setting the parameter in the Tor configuration file, and switched at runtime by sending a `SIGHUP` signal to Tor to reload its configuration file.

Chapter 5

Fine-tuning of the Algorithm

The parameter H in our algorithm determines how far back in time we want to look to calculate the cell counts for the circuits. This time horizon should be chosen to distinguish bursty HTTP circuits from circuits for continuous data transfer. For the PlanetLab experiments, the value of the parameter does not matter much, since the goal is to make HTTP circuits always have higher priority over bulk transfer circuits, and any value within a reasonable range will satisfy.

However, for the Tor nodes on the live network, the conditions are more complex. HTTP circuits are not only competing with bulk transfer circuits, they are competing with each other as well. The parameter should meet the requirement of distinguishing the two sets in practical scenarios. On the other hand, the standards may differ from OR to OR, however, depending on the capacity and the network condition. For example, if an OR is slow or H is set too small, the algorithm will quickly forget a circuit's past activity. A bulk transfer circuit will quickly drop to the same cell count as a newly created HTTP circuit, and compete with it. On the other hand, if the OR is fast or H is set too large, a newly created bulk transfer circuit will be prioritized over an HTTP circuit created long ago.

In this chapter, we experiment with different values of the parameter, to examine the effects of the parameter on HTTP traffic.

5.1 Testbed Setup

The testbed setup is similar to the setup of our live Tor network test. We selected a variety of parameter values for the middle OR, `planetgurgle`, and tested the performance.

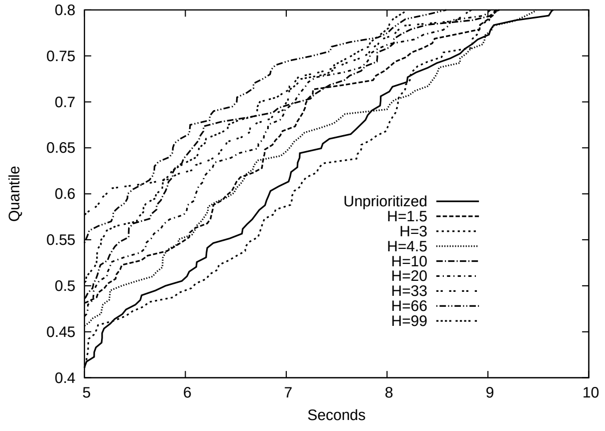

Figure 5.1: Comparison of performance for different values of H: CDF

We used git version 0.2.1.24 of Tor on the middle OR. In the configuration file, the parameter `CircuitPriorityHalflife` is the H value we mentioned earlier, which represents the interval after which the cell count for each circuit is decreased by half.

In this experiment, we randomly select a value for H for our middle OR from the set $\{-1, 1.5, 3, 4.5, 10, 20, 33, 66, 99\}$ (-1 indicates unprioritized), and fetch a small file hosted at the University of Waterloo. We repeat this until each value has 200 datapoints, and collate the results.

5.2 Experimental Results

The results for the download times for different values of H are shown in Figure 5.1 and Figure 5.2. Because of the density of the lines, we only show a fraction of the whole CDF in Figure 5.1. For ease of visualizing the data, in Figure 5.2, we show the 25th, 50th, and 75th percentile latencies for a range of different H values (the curves) as well as for unprioritized Tor (the horizontal lines).

The figures show that smaller values of H (1.5, 3, 4.5) perform only marginally better, if at all, than unprioritized Tor. This makes sense, as the past behaviour of a circuit will quickly be forgotten, and the bulk transfer circuits will have cell counts low enough to compete with HTTP circuits. The largest value 99 does not perform much better than unprioritized Tor either, since for our HTTP circuit, the behaviour will accumulate to a large cell count value and lose priority. The other values, 10, 20, 33 and 66, seem to be good values for our OR.

The results match our assumption: the H value only need ensure that bulk transfer circuits will have higher cell counts than HTTP circuits; a wide range of values can satisfy

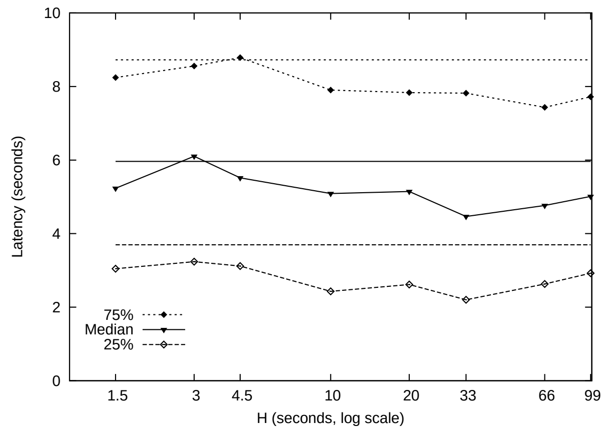

Figure 5.2: Comparison of performance for different values of H: latency vs. H. Values for unprioritized Tor are shown by horizontal lines.

this requirement. A value around 20 or 30 will likely satisfy most ORs in the Tor network. A global default value can be set in the directory authorities' consensus, so that OR operators do not need to manually configure it. Even if an OR operator misconfigures this value, for example, by setting it to 1.5, the performance will not be greatly harmed, as shown by our experiment.

However, as the Internet and the Tor network evolve, different protocols will start to use Tor, and the traffic distribution will not stay constant. The parameter should be regularly re-evaluated.

Chapter 6

Effects on Hidden Services

This chapter gives an overview of Tor's Hidden Services, and evaluates the effect of our algorithm on them.

6.1 Overview

Tor's Hidden Services [DMS04] allow users to provide TCP-based services, such as operating web servers, without revealing the server's IP address. A Tor client can both access and publish Hidden Services.

Figure 6.1 shows the structure of Hidden Services, where each arrow represents a three-hop connection over the Tor network, not a direct connection. We assume that the service provider Bob (represented by "Hidden Server" in the figure) wants to publish his hidden service. First, Bob generates a public/private keypair for his hidden service. Then he chooses a set of Introduction Points (IP) among the ORs, creates circuits to them, and sends introduction requests (Arrow 1). Those ORs will send back acknowledgement if suitable (Arrow 2), which means that they are ready to accept client requests.

After the set of Introduction Points is established, Bob publishes his service descriptor, containing the public key of the service and the list of Introduction Points, signed with the private key of the service, to the hidden service directory authorities, through another circuit to hide his identity (Arrow 3). After this step, the hidden service is published, and ready to be accessed by clients.

The *onion address* of the hidden service, which serves as a URL, is a hash of the service's public key. The advantage is that the address is self-authenticating: the users do not need to trust the descriptor downloaded from the directory servers; after connecting to

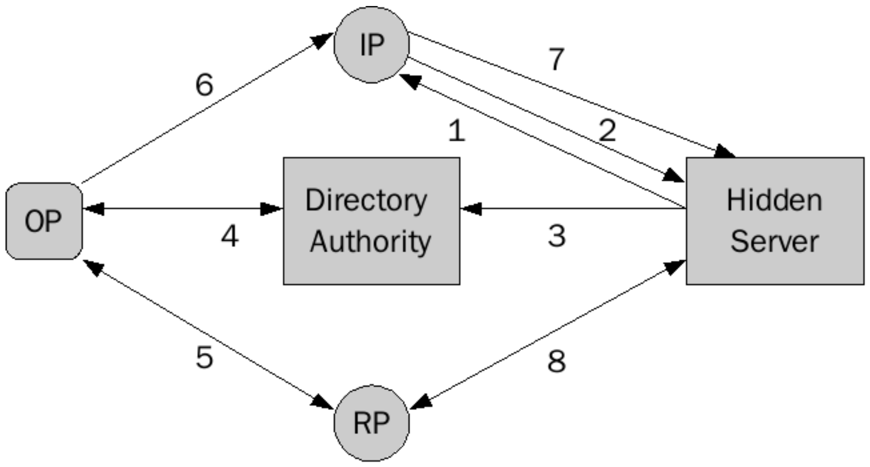

Figure 6.1: Overview of Hidden Services. Each arrow is a three-hop connection over the Tor network.

the service, she can authenticate the service's public key by the onion address. Bob then advertises the onion address to potential clients.

A client Alice (OP) wants to access Bob's hidden service. First she downloads the hidden service descriptor from the directory servers (Arrow 4). Then she establishes a Rendezvous Point (RP), selected from the set of ORs, for the data transfer later (Arrow 5). After receiving an acknowledgement from the Rendezvous Point, she contacts one of Bob's Introduction Points through a circuit, and tells Bob her Rendezvous Point through the Introduction Point (Arrows 6 and 7). Bob optionally authenticates Alice, and then connects to Alice's Rendezvous Point through a circuit (Arrow 8). The Rendezvous Point connects the two circuits, and Alice can attach application-level streams to the circuit, and access the hidden service.

Because of the different design goals, the underlying network traffic for hidden services is much more complex than for regular public services. Accessing hidden services involves more than a dozen ORs and multiple rounds of negotiation; as a result, it is much slower than accessing public services through Tor.

The most time-consuming part of the process for a client to access a hidden service is circuit creation. According to statistics [Loe08], the average time for hidden server circuit creation is 33.8 seconds, much higher than that of public services. After the circuit creation, the communication between the client and the server becomes almost as fast as accessing public services through Tor, as we will show later through our experiments.

Our EWMA algorithm is ideal for prioritizing the circuit creation phase of Hidden Services, since the command cells for circuit creation are the first cells sent by a circuit. These cells should get the highest priority almost independent from the value of the EWMA

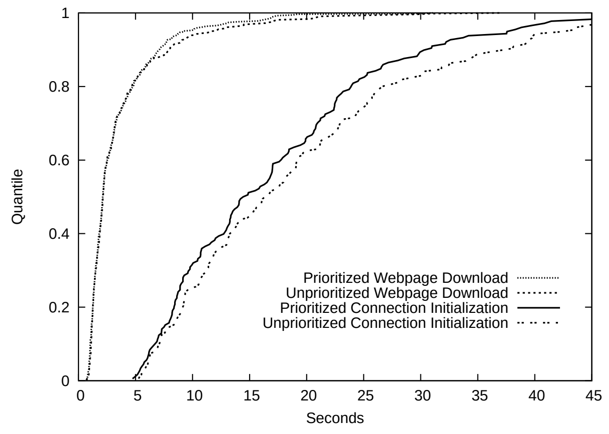

Figure 6.2: CDF for time cost of connection initialization and webpage fetching for Hidden Services, for unprioritized and prioritized Tor

parameter.

6.2 Experimental Results

We tested the performance improvement of Hidden Services, using a similar approach to our experiments in Section 4.3, above. We used $H = 33$ for our middle OR. The hidden server is a desktop machine at the University of Waterloo. The server is also configured to use `planetgurgle` as the middle OR for all of its circuits.

We modified webfetch to support SOCKS4a [Lee02], in order to resolve URLs for hidden services. We also instrumented webfetch to separately record the time for connection initialization and the time for webpage fetching. For each client instance, we perform four fetches of the target webpage. The latter three fetches will not include circuit creation, and so should be much faster than the first one. We performed 200 groups of tests (200 circuit creations). The connection initialization time for the latter three fetches is very short (averaging 2.0 seconds) compared to the first fetch. Since we are mainly concerned about the circuit creation, we only show the connection initialization time for the first fetch, and the download time for webpages. The CDF is shown in Figure 6.2.

As we notice, there are improvements in the connection initialization time in each quantile. The average time decreased from 19.3 seconds to 17.1 seconds (11.4%). This time consists of mostly circuit creation. Note that only two hops out of seven are prioritized (`planetgurgle`). We expect more improvement if all the ORs in the circuits are prioritized.

Chapter 7

Related Work

A number of works examine Tor in an attempt to improve its performance; we give an overview of some of them in this section.

7.1 Tor over DTLS

One area of investigation in improving Tor's performance is in fighting the improper application of TCP's congestion control mechanisms, which degrade Tor's performance. Between each pair of ORs, multiple circuits may share the same TCP connection, and their traffic is multiplexed within this connection using the same socket. When the number of unacknowledged packets in the socket buffer exceeds the socket's congestion window, TCP's congestion control mechanism takes effect, and TCP will refuse to send more packets until more acknowledgements are received. This mechanism is desirable if there is only one circuit using the connection; however, if there is more than one circuit, one circuit sending too much data and triggering congestion control will cause other circuits to stop sending as well.

Reardon and Goldberg [RG09] addressed this problem by using a TCP-over-DTLS tunnel. (DTLS — Datagram Trasport Layer Security — is the UDP-based analogue of the TCP-based TLS.) Instead of using a TLS/TCP connection between each pair of ORs, a DTLS/UDP connection is established, to prevent the congestion control mechanism incurred by one circuit from preventing other circuits from sending data. On top of DTLS/UDP, a user-level TCP connection is established for each circuit, to guarantee in-order delivery, congestion control and flow control on a per-circuit basis.

Reardon and Goldberg's work concentrated on a fair application of the congestion control mechanism: the fault of one circuit should not affect other circuits. In comparison, our approach aims to be fair on resource allocation among circuits: circuits that consume few resources recently should be prioritized over other circuits.

7.2 Opportunistic Bandwidth Measurement

Tor relies on the ORs' self-reported bandwidth values as weights to make router selection decisions. This is not necessarily accurate, and also may encourage malicious ORs to report a higher bandwidth to attract traffic. Snader and Borisov [SB08] proposed an opportunistic bandwidth measurement algorithm to replace the self-reported bandwidth; this is more accurate, and responds to changing load conditions quickly, while at the same time preventing low-resource routing attacks. In this way, Tor's bandwidth resources are allocated more efficiently, and the overall performance is improved.

Snader and Borisov also proposed a mechanism for users to tune Tor's parameter to select between circuits for higher performance or higher anonymity, while incurring very little cost of the other property.

7.3 Performance-Based Path Selection

Since the ORs are distributed globally, a circuit containing multiple intercontinental connections will inevitably bring higher latency. To prevent this, a user can configure her OP to optimize the path selection based on geographic location: she can choose the entry OR in her (client's) country, and choose the exit OR in the server's country, to minimize the latency incurred by cross country/continent links.

Panchenko et al. [PPR08] performed experiments on different policies of path selection: choose ORs uniformly randomly, from different continents, from the same continent, and from the same country (US and Germany). According to statistics, choosing ORs only from Germany gave the best results: the median time for circuit setup is 2.72 seconds, while choosing ORs from different continents resulted in the worst performance, as would be expected: the median time for circuit setup is 5.15 seconds. The results indicate that a low geographic diversity of the selected ORs will improve the performance in terms of latency.

Choosing ORs based on geographic location will harm the anonymity of the user, however: the middle OR knows that the client is likely from the same country as the entry OR, and the server is likely from the same country as the exit OR. In addition, choosing ORs from the same country will increase the risk of choosing ORs from the same operator, which further reduces anonymity. A geographic diversity of ORs will guarantee the security properties. Hence, a balance should be maintained between security and performance.

Geographic-location-based path selection is essentially attempting to minimize the link-wise latency. For a more accurate estimation, a user can actively send command cells to probe the link-wise latency, on which to base the path selection decision. Additional policies may be needed to guarantee geographic diversity.

7.4 Providing Incentives for Tor Relays

As pointed out by Dingledine [DM09], a main factor of Tor's performance issues is its capacity. There are too few ORs to support the current number of Tor users: only a tiny fraction of Tor users become ORs. With the increasing publicity of the Tor network, more ORs will join the network, but at the same time more users will enter the network to consume the increased capacity, making the OR/user ratio stable. Currently, there is no direct incentive for a Tor user to become an OR. Such incentives are needed to encourage more Tor users to volunteer as Tor relays and increase the OR/user ratio.

Direct financial support is also a good option for incentives. Androulaki et al. proposed a payment scheme, PAR [ARS+08], which pays e-cash to the relay operators. The evaluation indicates that the protocol incurs minimal overhead, but it relies on an e-cash infrastructure which is not currently available.

Alternately, one can give the ORs some kind of credit to allow them to perform faster when they themselves are using the Tor network as clients. Ngan et al. [NDW10] proposed such a reward scheme for Tor ORs. The scheme is based on the judgement of the trusted directory authorities to perform the bandwidth measurements of the ORs. If an OR's bandwidth exceeds some threshold value, the directory authority will give it a "gold star" in the directory listing. The traffic originated from a "gold star" OR will have higher priority at each OR along the path.

Since the measurement is performed by the directory authorities, an OR cannot cheat about its bandwidth to get a "gold star". It can, however, provide marginal performance to get the "gold star". Dividing multiple levels of performance reward based on an OR's bandwidth is not a good choice in terms of anonymity: it will result in an easier intersection attack by an adversary: based on the level of "gold star", an adversary can narrow the set of the initiator. Nonetheless, even this marginal increase in the bandwidth will greatly contribute to the overall capacity of the network.

7.5 Internet QoS Schemes

QoS refers to providing services that guarantee performance qualities for Internet applications. Those performance qualities include bandwidth, loss, delay and jitter. Several QoS schemes exist for today's Internet, including Integrated Services (RSVP), Differentiated Services, MPLS, and Constraint-Based Routing. [ZOS00] There are also schemes for prioritizing bursty traffic in ATM networks. In this section, we introduce some of them, and compare them to our work.

- Integrated Services

  Integrated Services is a QoS scheme which requires dynamic resource reservation. It negotiates with the routers in the communication path to reserve resources at each router, thus providing quantifiable QoS for the specified network flow. RSVP (Resource Reservation Protocol) is a protocol in which the receiver reserves resources along the path. Before the flow is initiated, the sender sends a PATH message to the intended receiver; along the path, each router reports its resource capacities in the PATH message. When receiving the PATH message, the receiver replies with a RESV message, requesting for resources along the path; routers on the path can either accept or reject the request. Once the request is accepted by all the routers, the resources such as bandwidth and buffer space will be allocated for the flow.

- Differentiated Services

  Differentiated Services attempt to divide services into several classes. It is a class-based service framework using packet tagging. The scheme uses the TOS byte in IPv4 header as Differentiated Service CodePoint (DSCP), to select the per-hop behaviour of the packet at each router. The per-hop behaviour specifies the relative quality received by the packet, such as weight for sharing bandwidth or priority for dropping. Before the packet enters the DSCP domain, its DSCP is defined by the client or the entry router according to the quality of service the packet should receive. Then, the router in the DSCP domain only need look at the DSCP field to decide the service for the packet. Compared to Integrated Services, no negotiation and reservation is required, thus providing better scalability.

- Queue Scheduling

  The end-to-end delay of a packet results from propagation, transmission and queueing delay. Propagation delay is the time the physical signal transfers between routers, and the transmission delay is given by the packet size divided by the link bandwidth. These two delays are determined physically. Queueing delay is the amount of time packets spent waiting in a queue before being transmitted. We can adjust queueing delay by adopting different scheduling policies. Basic policies include First Come First Serve (FCFS), priority scheduling, Weighted Fair Queue (WFQ), and others.

  This scenario slightly differs from Tor's ORs since ORs are not dedicated network routers. The Tor application only controls the order of cells within itself, not on the operating system level. Nonetheless, these scheduling policies still apply. For example, the cells in the OR's output buffer is sent out FCFS. If the cells can be tagged with priorities, a priority scheduling can be integrated in the output buffer to provide better QoS for prioritized circuits.

- Bursty Traffic Prioritization

  In [MF97], Fernandez et al. proposed an efficient scheduling policy to guarantee the performance of high-priority bursty traffic. Assuming that the connections at a switch share the outgoing bandwidth, the scheme divides the bandwidth resources into bandwidth units, each equal to the maximum burst of each connection. The connections are divided into two classes: high priority and low priority. Bandwidth units are allocated on the burst level: they will be allocated to a connection when the connection has a burst of traffic, and will be recycled when the burst is finished. High-priority connections are given the unit whenever a unit is available, while low-priority connections are given the unit only when the unit is available and the total number of low-priority connections is less than a threshold value. Once allcated the unit, the burst has exclusive use of the assigned bandwidth, and so the performance is guaranteed. With this scheme, a good balance of performance of connections and efficient use of bandwidth resource is achieved.

Our approach is similar to the above efforts, in the sense that Tor is an overlay network and ORs act like Internet routers; we wish to improve the QoS of HTTP traffic by adjusting the scheduling policies within ORs. However, none of them directly applies to our situation. For example, in contrast to Fernandez's scheme, our algorithm cannot decide beforehand which circuit should have higher priority: we aim to prioritize the bursts themselves, not a specified circuit. Other limitations exist as well: Tor does not have ISP support; deep packet inspection is unavailable since the packets are encrypted; in order not to violate Tor's availability, we cannot reserve bandwidth resources or drop excess circuits; due to Tor's decentralized design, payment for better performance is not yet available. These limitations prevent the existing Internet QoS schemes from directly applying to our situation.

Our EWMA algorithm is simple and effective in our situation. Nonetheless, incorporating ideas from other QoS techniques into Tor would be an interesting avenue for future work.

Chapter 8

Security Issues

Users choose Tor mainly for the security and anonymity it provides. Care should taken to make sure that any modifications to the architecture do not bring extra security vulnerabilities. In this chapter, we discuss the possible effects of our algorithm on the security properties of Tor.

8.1 Traffic Analysis

An important question is naturally whether our improvements in performance would enable an attack not previously present. We point out for emphasis that Tor is known to be insecure against an adversary that can see both ends of a circuit [LRWW04, MZ07, SS03]. The most apparent avenue for attack is for an attacker, who can see just one end of a circuit, to try to determine whether the other end is an OR that has been upgraded to use this protocol, or not, by observing the performance of the circuit.

Here, Tor's large variance in performance comes in handy. Although our method provides a noticeable improvement, the improvement is still small as compared to the very large variance; see Figure 4.9. It would seem to be as easy for an attacker to learn in the stock Tor, for example, whether the OR at the other end of the circuit had high or low bandwidth.

One may also contemplate an attack in the style of [EDG09], wherein the attacker constructs his own circuits through various ORs in the network in order to observe interference with a target circuit. This attack is already possible in stock Tor [EDG09]; the fixes to Tor made as a result of that paper (limiting circuits to eight hops) only prevent the bandwidth amplification portion of the attack.

8.2 Strategy Proofness

Since the prioritization decisions are based on the behaviour of each circuit, a user can modify her bulk transfer protocol to open many circuits and transfer parts of the file with each circuit in a bursty way. Each circuit will then have a lower EWMA value, and will be prioritized over HTTP and other interactive protocols.

Because our EWMA algorithm does not degrade the performance of bulk transfer a lot, there is no strong incentive for those bulk transfer users to implement such modifications. However, such modifications can make a specified protocol prioritized over any other protocol on Tor. We should note that our algorithm does not introduce this attack, since a user can still build multiple circuits on her protocol to make it more competitive, even with unprioritized Tor. Indeed, using multiple TCP connections on the regular (non-Tor) Internet will yield an unfair share of bandwidth. [PRG$^+$10] Investigations into the countermeasures will be a good direction for future work.

Chapter 9

Future Work

Our work concentrated on giving priorities to interactive streams in order to reduce the time an interactive cell spent within an OR, and thereby reduce the overall latency experienced by an interactive Tor client. There are other approaches that can prioritize interactive streams and reduce the time within an OR as well. In this chapter, we discuss those approaches and problems.

9.1 Reduce the Cell's Time in the Output Buffer

In the fine-grained analysis of a cell's time spent within the OR, we observed that our modifications resulted in a reduction in the amount of time a cell spends waiting to be flushed from the circuit queues of interactive circuits to the connection output buffer. We also observed that the cells still wait in the connection output buffer for a noticeable amount of time. If this time can be reduced, hopefully interactive circuits will benefit more in reducing the latency.

One possible approach is to make the output buffer a priority queue, instead of a FIFO list. Each cell will be tagged with a priority during the computation of the EWMA value, and the cells will be flushed from the output buffer in the order of priority. Since the number of cells is large, maintaining such a queue may bring large overhead as well.

A problem with this approach is that prioritizing individual cells instead of circuits will possibly result in priority inversion and head-of-line blocking: when the priority of a circuit changes from high to low, the order of cells belonging to this circuit in the output buffer will not change, but when the priority change from low to high, newly arrived cells will be tagged "high priority" and pushed to the output buffer, and will be inserted before those earlier arrived low priority cells. We should avoid this change of order so as to allow the exit OR to successfully regroup the data stream. If we tag the incoming high-priority cells as “low priority” when there are still low-priority cells from the same circuit left in the output buffer, the circuit will remain low-priority as long as not all low-priority cells are flushed at some point. This results in priority inversion.

Existing techniques can be adopted to avoid priority inversion in this situation. One simple solution is to upgrade the priority of all low-priority cells in the output buffer when a high-priority cell from the same circuit arrives. Then the original order of the cells is preserved. A more advanced data structure is required to implement the priority queue in order to ensure efficiency.

9.2 Prioritize Connections

By the protocol distribution in the Tor network [MBG$^+$08] and the bandwidth distribution among the ORs [Tor09], we can infer that only a small portion of interactive circuits get prioritized by our algorithm. Most of the interactive circuits will not be prioritized, although there are likely to co-exist bulk transfer circuits sharing the same OR that degrade the performance of interactive circuits. That is because although those circuits share the same OR, they do not share the same connection, thus interactive circuits cannot be prioritized within the connection.

Accordingly, another direction of future work is to prioritize connections on an OR as well. Simply assigning higher priority to the idle connections may be problematic, since a busy connection may not be doing a bulk transfer; it may simply contain many circuits, all doing web browsing. Slowing down this connection is not the desired behaviour.

Thus, besides watching the connection’s activity, we need to watch the circuits within it as well: namely, to count the number of interactive and non-interactive circuits within the connection, by using the EWMA algorithm, for example. Then, we can assign priority based on the number of interactive circuits within a connection: give higher priority proportionally to the connections with more interactive circuits. Along with the prioritization in the previous section, ideally, all the interactive circuits on an OR will be prioritized over those non-interactive ones.

Based on the EWMA algorithm, long term starvation for bulk transfer will not happen, since the cell count will drop exponentially when starved. However, compared to prioritizing circuits within a connection, prioritizing connections will possibly degrade the performance of bulk transfer: a connection with a single bulk transfer circuit will be deprioritized in this scenario, whereas it would not be in our existing scheme. Little’s Law may apply here, suggesting that the overall performance of bulk transfer would not degrade, but we require experiments to measure this effect empirically. On the other hand, a bulk transfer client will receive even better performance than in the unprioritized case when he creates multiple "interactive" circuits to perform the task.

Hence, we must ensure that any change in performance by prioritization should not be so drastic that it encourages bulk transfer users to strategically change their clients against the algorithm.

Chapter 10

Summary

In this work we examined one source of Tor's performance issues. One of the factors that contributed to the bad performance for interactive streams is the unfair scheduling algorithm between circuits: interactive circuits will be greatly slowed down because of co-existent non-interactive circuits on the same connection. We proposed an EWMA-based scheduling algorithm to prioritize the interactive circuits, and performed experiments on PlanetLab and the live Tor network. The results show that under realistic network traffic, interactive streams in prioritized Tor performs about 10% to 20% better, in terms of latency. The algorithm is completely compatible with the current Tor network: the ORs can be upgraded gradually, it can be turned on and off easily and on the fly, and will take effect immediately. Also, the algorithm brings little overhead, even on CPU-limited ORs.

References

[Aiu04] Tony Aiuto. webfetch. http://tony.aiu.to/sa/webfetch/, 2004. Accessed April 2010. 25

[Ano10] The Anonymiser. http://anonymizer.com, 2010. Accessed April 2010. 3

[ARS$^+$08] Elli Androulaki, Mariana Raykova, Shreyas Srivatsan, Angelos Stavrou, and Steven M. Bellovin. PAR: Payment for Anonymous Routing. In *Proceedings of the 8th Privacy Enhancing Technologies Symposium*, pages 219–236, 2008. 43

[Cha81] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. In *Communications of the ACM*, volume 24, pages 84–88, 1981. 3

[Dan03] George Danezis. Statistical Disclosure Attacks: Traffic Confirmation in Open Environments. In *Proceedings of Security and Privacy in the Age of Uncertainty*, pages 421–426, May 2003. 4

[DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Desion of a type III anonymous remailer protocol. In *Proceedings of 2003 IEEE Symposium on Security and Privacy*, pages 2–15, May 2003. 3, 5

[DM09] Roger Dingledine and Steven Murdoch. Performance Improvements on Tor or, Why Tor is slow and what we're going to do about it. http://www.torproject.org/press/presskit/2009-03-11-performance.pdf, 2009. Accessed April 2010. 9, 43

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004. 1, 3, 7, 37

[EDG09] Nathan Evans, Roger Dingledine, and Christian Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *Proceedings of the 18th USENIX Security Symposium*, pages 33–50, August 2009. 47

[Gra94] Volker Grassmuck. "Don't Try to Control the Network Because it's Impossible Anyway". `http://waste.informatik.hu-berlin.de/Grassmuck/Texts/remailer.html`, 1994. Accessed April 2010. 4

[GRS99] David Goldschlag, Michael Reed, and Paul Syverson. Anonymous Connections and Onion Routing. *Communications of the ACM*, 42(2):39–41, 1999. 3, 6

[GT96] Ceki Gulcu and Gene Tsudik. Mixing E-mails with Babel. In *Network and Distributed Security Symposium (NDSS 96)*, pages 2–16, February 1996. 3

[JAP06] JAP — Anonymity & Privacy. `http://anon.inf.tu-dresden.de/index_en.html`, 2006. Accessed April 2010. 3, 6

[Lee02] Ying-Da Lee. SOCKS 4A: A Simple Extension to SOCKS 4 Protocol. `http://ss5.sourceforge.net/socks4A.protocol.txt`, 2002. Accessed June 2010. 39

[LG08] John D.C. Little and Stephen C. Graves. Little's Law. `http://web.mit.edu/sgraves/www/papers/Little's%20Law-Published.pdf`, 2008. Accessed April 2010. 26

[Loe08] Karsten Loesing. *Privacy-enhancing Technologies for Private Services.* PhD thesis, University of Bamberg, 2008. 38

[Loe09a] Karsten Loesing. Measuring the Tor Network. `http://metrics.torproject.org/papers/directory-requests-2009-06-25.pdf`, 2009. Accessed April 2010. 9, 26

[Loe09b] Karsten Loesing. Measuring the Tor Network from Public Directory Information. `http://freehaven.net/~karsten/metrics/measuring-tor-public-dir-info-final.pdf`, 2009. Accessed April 2010. 1, 24

[LRWW04] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing Attacks in Low-Latency Mix-Based Systems. In Ari Juels, editor, *Proceedings of Financial Cryptography (FC '04)*, pages 251–265. Springer-Verlag, LNCS 3110, February 2004. 47

[Mat09] Nick Mathewson. gitweb.torproject.org. `http://gitweb.torproject.org/tor.git?a=commit;h=06e8370c33d6ccb73d55e9e8c3d2673c48d7b328`, 2009. Accessed April 2010. 30

[MBG$^+$08] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining Light in Dark Places. In *Proceedings of the 8th Privacy Enhancing Technologies Symposium*, pages 66–67, 2008. 10, 13, 20, 50

[MCPS03] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2. `http://www.abditum.com/mixmaster-spec.txt`, July 2003. Accessed April 2010. 3, 5

[MD05] Steven J. Murdoch and George Danezis. Low-Cost Traffic Analysis of Tor. In *Proceedings of 2005 IEEE Symposium on Security and Privacy*, May 2005. 8

[MF97] Matt W. Mutka and Jose Roberto Fernandex. A Burst-Level Priority Scheme for Bursty Traffic in ATM Networks. In *Proceedings of Sixth International Conference on Computer Communications and Networks*, pages 11–16, 1997. 45

[MZ07] Steven J. Murdoch and Piotr Zieliński. Sampled Traffic Analysis by Internet-Exchange-Level Adversaries. In *Proceedings of the 7th Privacy Enhancing Technologies*, pages 167–183, Ottawa, Canada, 2007. 47

[NDW10] Tsuen-Wan “Johnny” Ngan, Roger Dingledine, and Dan S. Wallach. Building Incentives into Tor. In *Proceedings of Financial Cryptography (FC10)*, January 2010. 43

[NIS10] NIST/SEMATECH. e-Handbook of Statistical Methods. `http://www.itl.nist.gov/div898/handbook/index.htm`, 2010. 28

[PACR03] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003. 18

[PPR08] Andriy Panchenko, Lexi Pimenidis, and Johannes Renner. Performance Analysis of Anonymous Communication Channels Provided by Tor. In *Proceedings of 2008 Third International Conference on Availability, Reliability and Security*, pages 221–228, March 2008. 42

[PRG$^+$10] Reinaldo Penno, Satish Raghunath, Vijay K. Gurbani, Richard Woundy, and Joe Touch. LEDBAT Practices and Recommendations for Managing Multiple Concurrent TCP Connections. `http://www.ietf.org/id/draft-ietf-ledbat-practices-recommendations-00.txt`, February 2010. Accessed April 2010. 48

[Rea08] Joel Reardon. Improving Tor using a TCP-over-DTLS Tunnel. Master's thesis, University of Waterloo, 2008. `http://uwspace.uwaterloo.ca/bitstream/10012/4011/1/thesis.pdf`. 14, 29

[Rea09] Joel Reardon. libspe. `http://crysp.uwaterloo.ca/software/`, 2009. Accessed April 2010. 21

[RG09] Joel Reardon and Ian Goldberg. Improving Tor using a TCP-over-DTLS Tunnel. In *Proceedings of the 18th USENIX Security Symposium*, pages 119–133, 2009. 41

[SB08] Robin Snader and Nikita Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *Proceedings of 16th Annual Network and Distributed System Security Symposium*, 2008. 42

[SS03] Andrei Serjantov and Peter Sewell. Passive Attack Analysis for Connection-Based Anonymity Systems. In *Proceedings of ESORICS 2003*, pages 116–131, October 2003. 47

[Tor09] Tor Network Status. `http://torstatus.kgprog.com/`, 2009. Accessed April 2010. 1, 8, 9, 10, 24, 50

[Web08] WebSiteOptimization.com. Average Web Page Size Triples Since 2003. `http://www.websiteoptimization.com/speed/tweak/average-web-page/`, 2008. Accessed April 2010. 18

[ZOS00] Weibin Zhao, David Olshefski, and Henning Schulzrinne. Internet Quality of Service: an Overview. `http://www.cs.columbia.edu/techreports/cucs-003-00.pdf`, 2000. Accessed April 2010. 44