Network Performance Improvements for Low-Latency Anonymity Networks

by

Mashael Saad Al-Sabah

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2013

© Mashael Saad Al-Sabah 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

While advances to the Internet have enabled users to easily interact and exchange information online, they have also created several opportunities for adversaries to prey on users' private information. Whether the motivation for data collection is commercial, where service providers sell data for marketers, or political, where a government censors, blocks and tracks its people, or even personal, for cyberstalking purposes, there is no doubt that the consequences of personal information leaks can be severe.

Low-latency anonymity networks have thus emerged as a solution to allow people to surf the Internet without the fear of revealing their identities or locations. In order to provide anonymity to users, anonymity networks route users' traffic through several intermediate relays, which causes unavoidable extra delays. However, although these networks have been originally designed to support interactive applications, due to a variety of design weaknesses, these networks offer anonymity at the expense of further intolerable performance costs, which disincentivize users from adopting these systems.

In this thesis, we seek to improve the network performance of low-latency anonymity networks while maintaining the anonymity guarantees they provide to users today. As an experimentation platform, we use Tor, the most widely used privacy-preserving network that empowers people with low-latency anonymous online access. Since its introduction in 2003, Tor has successfully evolved to support hundreds of thousands of users using thousands of volunteer-operated routers run all around the world. Incidents of sudden increases in Tor's usage, coinciding with global political events, confirm the importance of the Tor network for Internet users today.

We identify four key contributors to the performance problems in low-latency anonymity networks, exemplified by Tor, that significantly impact the experience of low-latency application users. We first consider the lack of resources problem due to the resource-constrained routers, and propose multipath routing and traffic splitting to increase throughput and improve load balancing. Second, we explore the poor quality of service problem, which is exacerbated by the existence of bandwidth-consuming greedy applications in the network. We propose online traffic classification as a means of enabling quality of service for every traffic class. Next, we investigate the poor transport design problem and propose a new transport layer design for anonymous communication networks, which addresses the drawbacks of previous proposals. Finally, we address the problem of the lack of congestion control by proposing an ATM-style credit-based hop-by-hop flow control algorithm, which caps the queue sizes and allows all relays to react to congestion in the network. Our experimental results confirm the significant performance benefits that can be obtained using our privacy-preserving approaches.

Acknowledgements

This dissertation is the product of four years of study, interactions and relationships with many wonderful and inspiring people, whose contributions made this dissertation possible. It is a great pleasure to give credit where credit is due. Before I do that, I would first like to thank Allah for all the blessings in my life. I have spent a lot of time and energy planning things, but Allah has always had nicer plans for me.

I would like to express my deepest appreciation and thankfulness to my supervisor, Ian Goldberg, for all his tremendous support, guidance, patience, motivation, contributions, support, encouragement, knowledge and skillfulness (and did I say support?). Ian accepted me as his student even though I had no previous research experience, and helped me develop the professional skills I need to conduct research and communicate my results orally, through talks, and in writing, through papers. Despite his busy schedule, Ian has always been very quick to provide help. I truly could not have wished for a better supervisor, and he is totally worthy of the world's best supervisor award. I would also like to thank Katrina Hanna for her kind words of encouragement regarding my work, accomplishments or talks.

I would also like to extend my appreciation to my thesis committee members, for their valuable time, feedback and suggestions. In addition, I am deeply indebted to Urs Hengartner for telling me about Tor in 2009, and for directing me to Ian. Thanks to Martin Karsten and Alfred Menezes for attending and providing feedback on my seminar talks. I am grateful to Michael Reiter for accepting to be my external committee member and for finding time in his schedule to travel and attend my oral defence in Waterloo.

I would like to thank my co-authors and other CrySP members for their contributions and support. I am thankful to Femi Olumofin, Tariq Elahi, Ryan Henry, Kevin Henry, Colleen Swanson, Sarah Harvey, and Roger Dingledine of the Tor Project for all the useful chats, discussions, feedback and words of encouragement regarding my work or talks. I am particularly thankful to Kevin Bauer for his great support and help. I have learned several research skills and benefited from his insightful opinions on my work. I truly enjoyed collaborating with him and Tariq.

Many thanks to my father, Saad Alsabah, and mother, Huda Alsayed, for their prayers, and unconditional love and support. I cannot find words to express my gratefulness to my sister, Noof Alsabah, and my mother, for travelling all the way from Qatar to Waterloo every year to help me overcome my occasional homesickness and for spoiling me with all the delicious food they cook for me. I am also very grateful to my best female friend, Amina, who always stood by me through the good and bad times.

I would like to thank my little family for helping me achieve my goals and for tolerating my absence and my moody days. Many thanks to my beloved husband and best friend, Khaled

Algarni, for the continuous emotional support, encouragement, understanding, care, effort and for always believing in me. Thank you for travelling with me so that I can attend conferences despite your commitments. Thank you for working very hard to make me happy, and thank you for giving me the push I needed to go back to work on my thesis whenever I felt desperate or slacked off. I would also like to thank my little angel, Turki Algarni, for cheering me up after receiving dreadful paper rejection emails, by saying "Don't be sad, Mommy." With you, sad times have been half as bad and happy times have been twice as good, and I am grateful to have you in my life.

Finally, I would like to thank the government of Qatar and Qatar University for the financial support, which made it easier for me to achieve my goals. I am particularly indebted to Dr. Sheikha Jabor Al-Thani, the Vice President and Chief Academic Officer at Qatar University, and Dr. Sheikha Al-Misnad, the President of Qatar University, for their encouragement, support, and advice. I am also very grateful to Dr. Ahmed Elmagarmid, the Executive Director of the Qatar Computing Research Institute, for always touching base with me and my progress, and for offering me a position among his team.

Dedication

To Khaled and Turki.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many Internet users believe that they are under the cloak of anonymity when the reality is that the Internet was born as an open research tool. In the online world, one's IP address is the digital fingerprint that links him to all his activities, and with the increasing trend of carrying out most of our daily tasks and activities, including mailing, social networking, shopping, voting and banking, online, we are revealing a significant amount of personal information every day, endangering our privacy. Still worse, there continues to be an escalating rate of privacy-related breaches and crimes everyday that range from occasional cyberbullying instances [CB12] to major incidents of governments losing sensitive data about thousands or even millions of their citizens [Gre06, BB07, Gre08].

Encryption can only solve part of the problem, as it can only hide the contents of messages people send online. Nevertheless, eavesdroppers can still observe who a person is conversing with, or what web server a user is connecting to. This can still reveal enough information to hinder one's privacy and online freedom. Encryption alone would not help a dissident wishing to publish anti-government documents online, for example.

Therefore, recent years have witnessed a dramatic increase in the use of *privacy-enhancing technologies* (PETs), as Internet users are gradually realizing the dangers of their privacy being violated. One popular emerging class of PETs, known as *anonymity networks*, allows users to perform their activities online anonymously. Anonymity networks are commonly *overlay networks*, logical networks built on top of the physical topology of the Internet. Users can maintain their anonymity by connecting to the Internet via paths established through the anonymity network.

One key anonymity network is Tor [DMS04], the most widely used privacy-preserving network with almost half a million users per day [To12b]. Not only does Tor enable its users to

maintain their privacy online, but it also provides them with a means to resist and circumvent network surveillance. Tor today is an influential anti-censorship technology that allows people in oppressive regimes to access information without the fear of being blocked, tracked or monitored. The importance and success of Tor is evident from recent global uprisings where the usage of Tor spiked [Din11] as people used it as a revolutionary force to help them fight their social and political realities.

Although Tor succeeded in attracting a rapidly increasing number of users since it was launched in 2003, the number of its volunteer-operated relays has not been growing at the same rate. Today, there are approximately 3,000 relays that run from all around the world [To12b]. Because of several design weaknesses in Tor, users experience poor performance that manifests itself in the form of large and highly variable delays experienced in response and download times during web surfing activities. In the next section, we elaborate on four key sources of poor performance in Tor, which we explore in this thesis.

1.1 Performance Problems in Tor

Since Tor is the most successful deployed anonymity network, it will serve as the basis anonymity network for our experiments and proposed improvements. Below we introduce four key causes of performance degradation in Tor.

1.1.1 Lack of Resources

Because Tor resources, such as its relay bandwidth and CPU, are provided by volunteers, they suffer from significant heterogeneity. For instance, the bandwidth capabilities of relays can range from as little as 20 KB/s to more than 20 MB/s. This results in high variability in performance as observed by the system users. This problem becomes even more evident with bandwidth-constrained relays. In Chapter 3, we discover that *bridges*—unadvertised Tor routers that provide Tor access to users within censored regimes like China—generally provide a lower quality of service than Tor's public infrastructure, and we propose and evaluate a dynamic traffic splitting scheme designed to improve the browsing experience for censored users. We also show that our traffic splitting scheme improves the performance of high-throughput applications such as video streaming.

1.1.2 Poor Quality of Service

Traffic congestion adds further delays and variability to the performance of the network. Previous studies have revealed that although Tor was originally designed for interactive applications such as web browsing and instant messaging, a small number of Tor users use bandwidth-greedy applications such as BitTorrent that consume a large and unfair fraction of the available bandwidth in the network [MBG+08].

In Chapter 4, we show how we use a novel machine-learning approach to classify Tor's encrypted traffic by application. This allows us to define different classes of service and thereby improve the experience of interactive application users.

1.1.3 Poor Transport Design

One key culprit to Tor's poor performance is its poor transport design. Tor multiplexes *circuits*, overlay paths established through the Tor network, from different users over the same TCP connection. Reardon and Goldberg [RG09] observed that since heavy circuits are often multiplexed with light circuits in the same TCP connection, and since heavy circuits have higher loss rates, they result in unfair application of the TCP congestion control of the shared connection on all circuits.

Our solution to this problem is presented in Chapter 5, where we propose and examine PCTCP, a novel anonymous communication transport layer design, in which every circuit is assigned a separate kernel-level TCP connection that is protected by IPsec, the standard security layer for IP.

1.1.4 Lack of Congestion Control

Although the original Tor design claimed to implement a congestion control algorithm, the reality is that the Tor network is actually not congestion controlled, but only flow controlled. Although flow and congestion control are often lumped together as one concept, they implement different functionalities. Flow control is mainly concerned with regulating flow in the network between two endpoints, so that the sender does not overrun the receiver. Congestion control, on the other hand, focuses on techniques that protect the network from congestion, a state in which a network node is overloaded because the rate of its incoming traffic is greater than the rate of its outgoing traffic.

TCP serves as a good example to distinguish the two concepts. To maintain flow control in TCP, a sender and a receiver negotiate a window size that controls how much unacknowledged

data a sender can forward to a receiver. The size of the window matches the transmission rate of the sender with that of the receiver. As for congestion control, TCP implements four different techniques: slow start, congestion avoidance, fast retransmit and fast recovery. In general, those techniques infer the congestion state of the network by utilizing timers and by monitoring acknowledgements in order to adjust the transmission rate according to the inferred conditions.

We visit the congestion control problem in Chapter 6 and explain Tor's current flawed approach to congestion control and show how we can design and implement an ATM-style hop-by-hop flow control to control congestion in the Tor network.

1.2 Anonymity loves company

Despite Tor's increasing popularity, the bitter reality is that it offers anonymity at the expense of intolerable performance costs. Not only do performance problems hinder Tor's wider adoption, but they can have an immense impact on its anonymity [DM06]. If users are discouraged from Tor's below-mediocre service, the anonymity set of all users would eventually shrink, which in turn reduces the anonymity guarantees obtained from the network today. Therefore, it is crucial to improve the performance and usability of Tor in order to enhance the anonymity it provides.

1.3 Goals

The aim of this thesis is to investigate how to improve the usefulness of low-latency anonymity systems by improving their network performance, which in turn boosts the anonymity provided to end users. We explore several techniques from the networking literature that aim to improve the performance of networks in general and tailor them to suit the privacy needs of anonymity networks. Our goal is to enhance performance, without weakening the threat model by introducing vulnerabilities or attacks, to the system in question.

Thesis statement It is possible to improve the performance of low-latency anonymous communication networks while maintaining the anonymity they provide to users.

Since Tor has become the *de facto* research platform for anonymous communication systems, we focus on Tor in the remainder of this dissertation for our proposed improvements and experiments. We explore the four main sources of performance degradation highlighted in Section 1.1, and propose solutions to address these problems. We evaluate our improvements consistently using networking metrics such as the network response and file download times. We also show that our techniques have a short road to deployment on the live Tor network.

1.3.1 Contributions

This thesis offers the following contributions to the area of low-latency anonymous communication systems:

- **Increasing throughput and enhancing load balancing.** In Chapter 3, we explore the performance benefits of utilizing multipath routing and traffic splitting for low-latency anonymous communication systems. Using Tor as a testbed, we find that our technique can significantly improve the experience of web browsing bridge users. This is important as bridges tend to be more resource-constrained than the rest of the network. We also find that our techniques can significantly improve the experience of high-throughput applications such as video streaming applications.
- **Improving Quality of Service** In Chapter 4, we enable differentiated QoS using network traffic classification in anonymous communication systems. We use a machine-learning approach to classify Tor's encrypted circuits in real time into application categories. This allows us to define appropriate QoS rules for each application. We show that our classification approach allows us to substantially improve the experience of interactive application users, which currently get an equivalent service to other classes of applications, such as the bulk applications, despite the fact that Tor was designed for real-time interactive applications.
- **Improving the transport layer design** In Chapter 5, we explore an alternative transport design for anonymous communication networks that addresses the shortcomings of the existing transport layer designs. In our design, instead of multiplexing circuits in the same TCP connection between any two relays, we dedicate a separate TCP connection to each circuit, and protect the IP layer using IPsec. We show that our design provides significant performance benefits for interactive application users.
- **Improving congestion control.** In Chapter 6, we identify the problem of the lack of congestion control in Tor, and provide a fresh approach to congestion and flow control inspired by techniques from ATM networks. We implement a per-link credit-based flow control algorithm called N23 [KBC94] that allows Tor routers to explicitly bound their queues and signal congestion via back-pressure. We also show that our technique succeeds in reducing unnecessary delays and memory consumption.

1.4 Outline

The remainder of this thesis is organized as follows. Chapter 2 provides a brief background on the area of anonymous communication, and surveys previous proposals that aim to improve

low-latency anonymous communication systems. Chapter 3 investigates the use of multipath routing in anonymous communication systems and illustrates its benefits to bridge and video streaming users. Chapter 4 introduces QoS to anonymous communication networks as a means to provide interactive application users with the higher responsiveness they need from the network. Chapter 5 studies and presents an alternate transport layer design for anonymous communication systems that addresses the shortcomings of existing designs. Chapter 6 explores the problem of the lack of congestion control in Tor and proposes a credit-based algorithm inspired by ATM networks. Finally, we conclude in Chapter 7.

Chapter 2

Background and Related Work

Since the introduction of mix-nets in 1981, the area of anonymous communications has evolved into two streams, based on their design and the scope of applications they support: the low-latency and the high-latency anonymity systems. In this chapter, we start by giving a brief overview of anonymous communications. Next, since we focus on the Tor network as our research platform, we provide a detailed background on Tor, and survey previous proposals that aim to improve the performance of the Tor network.

2.1 Anonymous Communication Systems

The goal of anonymous communication is to solve the traffic analysis problem, which Chaum defines as follows: "The problem of keeping confidential who converses with whom, and when they converse." [Cha81] To that end, anonymous communication systems are designed so that users in the system communicate with their destinations through a single or a number of intermediate hops, where every hop only knows the next and previous hops. Therefore, no hop alone can link the sender with the receiver (unless only a single intermediary hop is used). Messages relayed in the system are generally fixed in size, and they are cryptographically altered at every hop.

2.2 Low-latency Anonymity Systems

Figure 2.1 summarizes the comparison between low-latency anonymity systems and their high-latency counterparts. High-latency anonymity systems, like Mixminion [DDM03], and mix-

	Threat model	Example Systems	Example Applications
Low-latency	Weaker adversary (local and active/passive)	Tor Anonymizer	Browsing Instant Messaging SSH
High-latency	Well-funded adversary (global and active/ passive)	Mix-net Mixminion	Email E-voting

Figure 2.1: Comparison between low-latency and high-latency anonymity systems

nets [Cha81] assume a powerful *active global* adversary—one which is able to monitor the input and output links of every node, usually called a mix, in the network. To hide the correspondences between the incoming and the outgoing traffic of a mix, equal-length messages are shuffled, cryptographically altered, and stored for some intentionally added delay before they are sent to the next mix or destination. Because of the additional delay, which can be up to several hours, high-latency anonymity systems can only support delay-tolerant applications, such as e-voting and email.

Both high- and low-latency anonymity systems, such as Tor and Anonymizer [Ano], also assume an *active adversary*, one that can add, delete, or delay traffic. In addition, low-latency anonymity systems assume a more relaxed threat model: an active partial adversary who can monitor part of the network (no more than 20% of the nodes, for example). This class of anonymity networks is designed to support interactive applications like instant messaging, web browsing and SSH connections. Next, we present a detailed background on Tor, the most widely used anonymity network, and survey previous proposals that aim to improve its performance.

2.3 Tor

Tor is a low-latency anonymity network that is based on the concept of onion routing [RSG98]. The network today consists of approximately 3000 volunteer-operated relays [To12c], known as

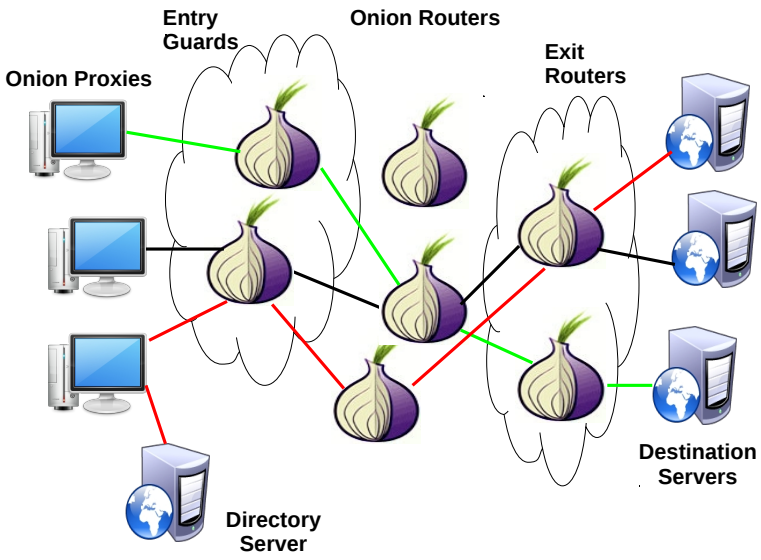


Figure 2.2: The Tor network

Onion Routers (ORs). Each OR creates a *router descriptor* that contains its contact information, such as its IP address, ports, public keys, and its bandwidth capabilities, and sends the descriptor to *directory authorities*. Tor clients, nicknamed *Onion Proxies* (OPs), download the router descriptors from directories to build paths, referred to as *circuits*, through the network before they can communicate with their Internet destinations. Each circuit usually consists of three ORs, which are referred to as the *entry guard*, *middle*, and *exit* OR, according to their position in the circuit. ORs in a circuit are connected by TCP connections and TLS [DR08] is used to provide hop-by-hop authenticity, data integrity and confidentiality. Traffic in Tor travels in fixed-sized units (512 bytes) called *cells*. Figure 2.2 visualizes the Tor network.

2.3.1 Circuit Construction

For performance reasons, an OP preemptively creates a number of spare circuits for its user applications. When the OP receives a new TCP stream from the user application, it attaches it to an appropriate pre-established circuit. If no such circuit exists, the OP builds a new circuit by first selecting three routers, X_i, according to Tor's bandwidth-weighted router selection algorithm (described in Section 2.4.2). Next, to start establishing the circuit, the OP sends a *create_fast*

command to X_1, which responds with a *created_fast* reply. To extend the encrypted channel, the OP sends an *extend* command to X_1, containing in its payload a *create* command and the first half of a Diffie-Hellman handshake for router X_2, encrypted to X_2's public key. Router X_1 forwards this *create* command to router X_2, and when it receives a *created* cell back from router X_2, it forwards its payload in an *extended* cell to the OP to finish the client's DH handshake with router X_2. The same procedure is carried out for each subsequent OR added to the circuit.

2.3.2 Circuits and Streams

The client can multiplex several TCP *streams* over one circuit, which generally has a lifetime of ten minutes. The client uses a circuit as follows: The client's web browser must be configured to use the OP, which uses a SOCKS proxy to listen for incoming browsing traffic. The OP divides traffic to fixed-sized cells, adds a layer of encryption for every node on the forward path and then cells are source-routed through the established circuits. Every hop, on receiving a relay cell, looks up the corresponding circuit, decrypts the relay header and payload with the session key for that circuit, replaces the circuit ID (a unique number that distinguishes circuits between any two routers) of the header, and forwards the decrypted cell to the next OR. When the exit OR receives the cell, it removes the last layer of the onion encryption, and establishes the connection on behalf of the user to the intended destination. Therefore, only the exit node can observe the user's traffic, but only the entry guard knows the identity of the user. If both the entry guard and exit node cooperate, they can use traffic analysis to link the initiator to his/her destination.

2.3.3 Circuit and Stream Windows

Tor uses two layers of end-to-end window-based flow control between the circuit end points (the exit and the OP) to ensure that the sender does not overrun the receiver's input buffer. First, a *circuit window* limits how many cells may be in flight per circuit. Tor uses a fixed 500 KiB (1000 cell) circuit window, meaning that only 1000 cells can be traveling through the circuit at any time. When an end point (OP or an exit) sends a data cell through the circuit, the size of the window is decremented. For every 50 KiB (100 cells) received, an acknowledgment cell called the `SENDME` is sent from the other end, to inform the sender that they may increment the window by 100 and can forward a number of cells that is equal to the current circuit window size.

Within each circuit window is a *stream window* of 250 KiB (500 cells) to provide flow control for streams within a circuit. The receiver replies with a stream-level `SENDME` for every 25 KiB (50 cells) received. On receiving a stream-level `SENDME`, the sender increments the window size by 50 and can forward 50 more cells.

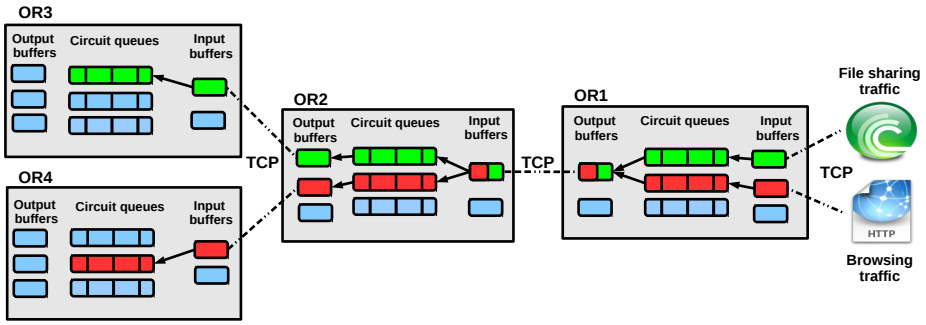


Figure 2.3: The cross-circuit interference problem: the figure demonstrates the cross-circuit interference problem when a single TCP connection is shared between a loud and a quiet circuit. OR1, acting as an exit for both circuits, receives file-sharing data and web browsing data on two different connection input buffers. The cells then are pushed to their circuit queues. Since the next hop for each circuit is OR2, both circuits share the same connection output buffer. Since the file-sharing circuit is expected to drop more data on the connection between OR1 and OR2, the web browsing circuit experiences more delays due to the unfair application of TCP congestion control on the shared connection.

2.3.4 Circuit Multiplexing

Tor's OPs and ORs communicate with each other using TCP connections. Every OR-to-OR TCP connection multiplexes circuits from several users. Reardon and Goldberg [RG09] pointed out that this design can potentially hinder the performance of interactive circuits. This problem is illustrated in Figure 2.3. The connection between OR1 and OR2 in the figure depicts a scenario where a noisy circuit, carrying BitTorrent traffic for example, is multiplexed with a circuit carrying interactive web browsing traffic. In this case, TCP congestion control would be unfairly applied on both circuits whenever the noisy circuit triggers congestion, due to lost or dropped packets, on the shared TCP connection. Since the amount of data transmitted by file sharing applications is significantly larger than that by interactive applications, it is expected that bulk application circuits trigger congestion control more often than interactive circuits. However, TCP congestion control would apply on all circuits equally and would result in extended queuing times for data cells in TCP output buffers and thereby, longer delays observed by clients.

2.3.5 Tor's Queuing Architecture

Tor uses a tiered buffer architecture to manage cells traveling through circuits, as also shown in Figure 2.3. When an OR receives a cell from an external server or from another OR or OP, the cell is passed from the kernel TCP receive buffer to a corresponding 32 KiB connection-level input buffer in Tor. After the cell is encrypted or decrypted, it is placed on the appropriate FIFO circuit queue. Since several circuits share the same connection output buffer, a scheduler is used to retrieve cells from the circuit queues to be placed on a 32 KiB output buffer. Finally, the cells are sent to the kernel TCP send buffer which flushes them to the next OR or OP.

2.3.6 Circuit Scheduling

Tor uses a label-switching design that multiplexes several circuits across the same Tor routers. In order to ensure that each circuit is given a fair share of the routers' bandwidth, Tor employs a round-robin queuing mechanism. Each circuit is serviced in a first-come, first-served manner, which ensures that each circuit is given a fair share of the available bandwidth.

However, McCoy *et al.* [MBG+08] have shown that the distribution of application traffic on Tor is not uniform across all circuits: a relatively small number of circuits (e.g., bulk file downloaders) consume a disproportional amount of the network's bandwidth. To mitigate the unfairness, Tang and Goldberg [TG10] proposed a circuit scheduling prioritization scheme (described in Section 2.4.1) so that interactive circuits tend to be serviced before bulk-downloader circuits. This prioritized circuit scheduling is currently deployed on the live Tor network.

2.3.7 Bridges

Beyond enabling anonymous communications online, Tor has become an essential tool in circumventing Internet censorship. Today, regimes around the world continue to aggressively filter [XMH11], monitor [PKK08], or explicitly block access to certain types of online content. While Tor offers online privacy and censorship resistance to hundreds of thousands of users on a daily basis, its public infrastructure of public relays can be easily blocked. In response, Tor uses special unlisted relays called *bridges* to aid users residing within regimes, such as China, that explicitly block the Tor network. Clients learn about bridges by visiting `https://bridges.torproject.org` or by sending mail to `bridges@bridges.torproject.org` from a Gmail account. Clients usually get a response consisting of the contact information of three bridges. Clients can then configure their OPs to use the bridges they learn about as first hops in their circuits.

2.3.8 Tor's Threat Model

Anonymity is maintained for Tor's users because only the entry OR receives a direct connection from a user, and only the exit OR forms a direct connection to the destination. Therefore, no single entity can link users to their destinations. The threat model in Tor assumes a local active adversary that can watch or control part of the network (no more than 20%) and can add, delete or modify traffic in the network, as explained in Section 2.2. The anonymity of a Tor circuit is compromised if the adversary can watch the two ends, the entry and exit, of the circuit.

2.4 Previous Proposals to Improve Anonymous Communication Systems

Research on improving anonymous communication in general and Tor in particular has evolved into five main streams: reducing network congestion, router selection, scalability, circuit construction, and security. In this section, we start by providing an overview of a number of proposals aimed at improving the performance and security of Tor. First, we start by describing several proposals that aim to reduce congestion using an alternate transport layer design or scheduling. Second, we overview the path selection algorithm in Tor, and some work proposed to achieve better routing. Also, we briefly describe the circuit construction mechanism of Tor and some key proposals that aim to improve it. Finally, we present three categories of attacks that have shown their effectiveness against Tor.

2.4.1 Relieving Network Congestion

TCP over DTLS. In his thesis [Rea08], Reardon studied the effects of TCP congestion control on the performance of Tor. He found that because Tor multiplexes several streams of data on the same TCP connection, this results in an unfair application of TCP's congestion control protocol. First, the nature of TCP's congestion control mechanism results in multiple data streams competing to send data over a TCP connection that gives priority to circuits that send more data. A busy stream that triggers congestion control will cause low-bandwidth streams to struggle to have their data sent. Second, when packets are dropped or reordered from one circuit, all other circuits going through the same TCP connection are penalized. The reason is that the TCP stack will buffer available data on input buffers until the missing in-order component is available. His solution is to use TCP-over-DTLS (Datagram Transport Layer Security) transport between routers, and give

each stream of data its own TCP connection, while hiding from observers which packets belong to which TCP connections using DTLS.

BRAIDS [JHK10]. The goal of BRAIDS is to give clients incentives to run as relays. The incentive is improved performance. Clients first obtain free tickets from a bank, which is a centralized and partially trusted offline entity that is responsible for bandwidth accounting tasks. Clients remain anonymous, as they use blind signatures to get the signed tickets from the bank. The tickets are relay specific, which solves the problem of double spending; this means that no client can use a ticket twice. When a client wishes to get improved service from a relay R, it can present it with tickets. Each ticket allows the client to receive an improved prioritized service for a fixed number of data cells. After the ticket is used, the circuit priority is lowered, but can be restored after a new ticket is presented. A used ticket is utilized by R as a voucher that is redeemable for a new relay-specific ticket for another relay C. Therefore, the more service a relay R provides, the more tickets for improved service it can collect. Experimental results showed that the more clients convert to relays, the better overall performance is achieved for both file-sharing clients and web browsers.

The Gold Star scheme [NDW10] aims to incentivize Tor clients to relay anonymous traffic. Trusted authoritative directories are responsible for assigning "gold stars" to relays after evaluating their bandwidth capabilities, and after testing their faithfulness in relaying traffic. A gold star's relay traffic is given higher priority by other relays, which means they always get relayed ahead of other traffic. In this scheme, there are only two types of traffic: gold star prioritized, and other non-prioritized. The reason for the simple two-level classification of the proposed scheme is to protect the anonymity of users that can be hurt if an observer is able to reduce the anonymity set based on the class of service of a circuit. This scheme is very simple and easy to implement in the current Tor network. Experimental results have shown that cooperative users—clients that donate an amount of bandwidth to relay network traffic—witness significant improvements in download times and ping times, even under heavy background traffic. Also, cheating users that stop relaying network traffic after achieving a gold star status are penalized with degraded performance whenever they toggle to their selfish behaviour.

Improved Circuit Scheduling [TG10]. Because Tor multiplexes several circuits in a single connection, interactive traffic circuits are crowded out by bulk file sharing circuits. This results in degraded responses for interactive traffic, which is unacceptable because unlike for bulk traffic, low latency is a key requirement for interactive traffic. To solve that problem, the authors propose prioritizing circuits that have interactive traffic. Because it is hard to identify the type of traffic in an anonymity network, the insight of this work is to change the scheduling algorithm so that it gives precedence to circuits that have sent fewer cells recently, thereby increasing their responsiveness. To achieve that, each circuit maintains a state variable that keeps track of the exponentially weighted moving average (EWMA) of the number of cells sent. A smaller such

value is likely to identify a circuit with interactive traffic, which the scheduler prioritizes and gives service next, as opposed to Tor's original round-robin scheduling. Small-scale live experiments show that this technique does not hurt the performance of bulk transfer circuits. However, later work by Jansen *et al.* [JH12] showed that the performance of bulk clients can be noticeably worse when this prioritization algorithm is used. Also, web clients only benefit from the EWMA circuit scheduler under heavy traffic loads, but performance degradation might be experienced under light network loads.

2.4.2 Router Selection

Tor's Router Selection Algorithm. In the original Tor proposal, ORs are selected uniformly at random for circuit construction. However, due to user heterogeneity, the algorithm was later changed to fulfill the following constraints:

1. No router appears more than once on a path, and no two routers in the same circuit belong to the same class B network (/16 subnet) or the same *family*. Co-administered routers can be marked as belonging to the same family by operators to avoid hindering users' privacy.

2. As of 2006, to defend against some attacks, such as the predecessor [WALS04] and locating hidden services [ØS06] attacks, Tor's router selection algorithm was changed so that the entry node is chosen from a subset of nodes known as the entry guards. An entry guard is a node whose weighted fractional uptime is at least the median for active routers, and its bandwidth is at least the median or at least 250KB/s [DM13]. Currently, guards selected are assigned a validity period of 30–60 days, and used throughout that time for all circuits.

3. Selecting a subsequent OR on the path is proportional to its offered bandwidth, in order to ensure that more capable routers are chosen more often. If b_i is the bandwidth offered by router i, then router i is chosen with probability $b_i / \sum_{k=1}^{N} b_k$, where N is the total number of routers in the network.

Snader and Borisov [SB08] identified two key weaknesses in this design. First, the bandwidth is self-reported. This allows some routers to misbehave by reporting exaggerated bandwidth capabilities in order to lure more circuits to use them [BMG+07], in order to increase the rate of compromise. Even when honest bandwidth values are reported, they are still a poor indicator of the available capacity because of the network dynamics and congestion state. Second, it does not provide users with the flexibility to trade off anonymity with performance according to their requirements.

Therefore, to solve the self-reported bandwidth problem, they proposed an *opportunistic bandwidth monitoring* approach, where every router aggregates the bandwidth capabilities of other routers it contacts over time, and then reports these measurements to the authorities. Also, they introduce *Tunable Tor*, an algorithm that allows users to configure the level of performance they want to trade off with anonymity. Briefly, Tunable Tor works as follows: a list of ORs is sorted according to some criteria (such as the opportunistic bandwidth measurement). If this list is indexed from 0 to $n-1$, then the router selected is that with the index $\lfloor n \cdot f_s(x) \rfloor$, where x is selected uniformly at random from $[0, 1)$. f_s is a family of functions $f_s : [0, 1] \to [0, 1]$ given by:

$$f_s(x) = \begin{cases} \frac{1-2^{sx}}{1-2^s}, & s \neq 0 \\ x, & s = 0 \end{cases} \tag{2.1}$$

Configuring a higher value for s results in a selection prejudice towards routers with higher ranking in the list. If $s = 0$, the router is chosen uniformly at random.

Murdoch and Watson [MW08] compared the performance of four different Tor path selection algorithms: the original uniform relay selection, Tor's current bandwidth-weighted relay selection, and Tunable Tor with the minimum suggested s for improved anonymity ($s = 1$), and the maximum suggested s for improved performance ($s = 15$). In their evaluations, they used two performance metrics: probability of path compromise and network latency. They used queuing theory to model the latency expected with the different path selection algorithms. Their latency results demonstrated that Tor's weighted bandwidth selection algorithm provides improved performance over the other router selection algorithms. Moreover, it also showed improved anonymity against a node-rich and bandwidth-poor attacker. The reason is that when higher-bandwidth nodes have a higher probability of being selected, the algorithm deviates further from selecting malicious poor-bandwidth nodes.

Sherr *et al.* [SBL09] note a number of problems in the opportunistic bandwidth monitoring described above. Routers reporting the bandwidth of other monitored routers can lead to two undesirable effects. First, routers can lie about the bandwidth of other routers they report. If a router is cooperating with other malicious routers, then they can report exaggerated bandwidth capabilities about members of their coalition. This problem, however can be addressed using Eigenspeed [Sna10], which is an opportunistic bandwidth measurement algorithm that is resilient in the face of malicious attacks. The second problem that Sherr *et al.* point out is that for every router to report the performance of other routers contacted means revealing information about established circuits, giving the servers a more powerful global view of the network. Alternatively, they propose replacing the opportunistic measured bandwidth in Tunable Tor with a *link-based* metric. Their observation is that choosing paths based on link characteristics such as latency, jitter, or number of traversed Autonomous Systems can provide improved performance over

node-based characteristics alone. Their proposed link-based Tunable Tor takes place in two phases. In the first phase, the initiator generates various candidate paths, and then the end-to-end cost of each path is computed according to the desired link-based metric. In the second phase, paths are sorted according to their metric, and the Tunable Tor algorithm is used to trade off between performance and anonymity.

One assumption made in the link-based router selection algorithm is that the initiator must maintain knowledge of the costs of the whole network, in order to be able to compute the cost of the whole path. For example, if a user wishes to use latency as a metric in constructing circuits, then it must measure the pairwise latency between every two routers in the network. The cost of this measurement can outweigh the benefits of exploiting link-based metrics. For that reason, Sherr *et al.* also propose the use of a *network coordinate system*, a multi-dimensional space in which the distance between relays in the virtual coordinate corresponds to the metric utilized in the router selection algorithm.

Multipath Routing in Tor. Multi-path routing has been studied in the context of onion routing by Snader [Sna10]. In this work, Snader simulated downloading a 1 MB file over a Tor network simulator. The file was divided into 512-byte chunks and sent over multiple circuits simultaneously. Throughput is significantly improved with the use of multiple circuits; however, using two circuits performs better than using one circuit or more than two circuits (using more than two circuits increases the chances of choosing a slow router). The median transfer time remains unchanged for more than two circuits, but the 90^{th} percentile transfer times actually increase when the number of circuits used is greater than two. From a security point of view, the anonymity of a user can be compromised if all paths are monitored, a case that assumes the adversary controls a large fraction of the network, which is outside the threat model of Tor. Since we also explore multipath routing and traffic splitting in Tor, we contrast our approach with Snader's approach in Section 3.7.

LASTor. Recently, Akhoondi *et al.* [AYM12] proposed a client-side router selection algorithm called LASTor which exploits the geographical location of Tor routers in order to minimize latencies observed by clients. The router selection approach uses a tunable weighted shortest path (WSP) algorithm that allows clients to trade off anonymity and performance. LASTor also protects clients from observers at the Autonomous System (AS) level, as it implements a lightweight technique to reliably avoid creating paths that have the same AS on the path between the client and its entry guard and the path between the exit and the client's destination.

Wang *et al.* [WBFG12] recently proposed a path selection algorithm that uses latency as an indicator for circuit congestion. First, Tor's default bandwidth-weighted router selection algorithm is maintained to build circuits. Then, the proposed algorithm uses opportunistic, as well as active probing techniques, to obtain latency measurements. The client remembers the latency measure-

ments for the individual relays which can be useful in two ways. First, if a client is using a congested circuit, it can switch to a less congested circuit. Second, the router selection algorithm is also modified to take into account the latency measurements, in addition to the bandwidth, for candidate relays chosen to build circuits.

Evaluation of Router Selection Algorithms. Wacek *et al.* [WTBS13] evaluate the performance and security of all router selection algorithms described above. The authors use an emulated scaled-down Tor network that realistically models the live Tor network. The authors then implement the router selection algorithms and compare their performance in terms of throughput, time-to-first-byte and average ping times. They also evaluate the anonymity of these algorithms using the Gini coefficient, entropy and compromise rate due to the AS-level adversary. The evaluation shows that the congestion-aware algorithm proposed by Wang *et al.* outperforms other router selection algorithms without reducing anonymity. LASTor, on the other hand, provided the poorest performance among other algorithms, but maintained high anonymity guarantees.

2.4.3 Improving Scalability

Torsk [MTHK09] investigates the router selection problem from a different angle. Torsk seeks to address the scalability problem inherited from the centralized-bootstrapping design in Tor, in which clients are required to download the network status information from the directory authorities periodically, imposing a significant bandwidth cost on the network. The authors point out that with the growing number of Tor users, the network in the near future will spend more bandwidth in bootstrapping than in providing actual anonymous communication. To address this challenge, Torsk proposes a *decentralized circuit construction* scheme that uses a combination of a DHT (distributed hash table) structure for bootstrapping, and a *buddy selection protocol* for peer discovery.

To perform bootstrapping, Tor's directory authorities are in Torsk given the role of the Neighbourhood Authority (NA), an entity responsible for issuing certificates to neighbors in the DHT space when nodes join or leave the network. This allows bootstrapping to be a low-cost operation, as new nodes are only required to generate a new ID and perform a lookup on their ID to find the closest neighbor, as opposed to downloading the whole topology information. Next, the newly joining node contacts the NA, which takes the ID of the node and the certificate of its closest neighbor to generate a new certificate for all affected neighbors.

Router selection is carried out as follows. First, every relay uses the buddy selection protocol to find other random relays to consult during DHT lookups. Likewise, a client uses the buddy selection protocol to begin a random walk in the network to discover an entry guard. Next, for subsequent relays, the client uses the furthest OR on the partially constructed circuit R_i

to randomly find R_{i+1} by consulting R_i's previously found lookup buddies. Finally, because lookups are loud and can reveal information about the constructed circuit, cover traffic is also used to add noise in case lookups are profiled.

PIR-Tor [MOT+11] steps away from the P2P paradigm to address the scalability problem in Tor. Instead, it advocates for the use of Private Information Retrieval (PIR) techniques in order for clients to be able to download a fraction of the network view without hindering their anonymity. The goal of PIR is to allow a client to retrieve a record from the database, without revealing that record to the server. The current approach in Tor is for clients and relays to download the whole database (the trivial PIR solution) and then choose their desired records, a costly operation and a main reason for the scalability problems in Tor. The advantages of PIR-Tor over P2P designs is twofold. First, the client-server bootstrapping architecture of Tor is preserved making for an easier deployment path. Second, the security guarantees of the system are easier to analyze than former P2P designs.

The authors investigated two flavors of PIR techniques: computational PIR (CPIR) and information theoretic PIR (ITPIR). For CPIR, some relays are selected to act as the PIR servers. To build a circuit, a client contacts one of the CPIR servers to perform two PIR lookups: one for a middle relay and another for an exit relay. On the other hand, since ITPIR requires the use of multiple servers (the privacy of a user's query is guaranteed if a threshold number of the ITPIR servers do not collude), ITPIR server functionality can be implemented between a client and its entry guards. This reduces the PIR lookup for each circuit to only one to lookup an exit node, while a middle node can be retrieved by a normal lookup. Evaluation of PIR-Tor showed that both techniques of PIR help reduce the communication overhead as the network scales. However, only ITPIR provides the same level of security of the current Tor network, as CPIR requires fewer lookups to scale, which means clients have to reuse the retrieved relays in building several circuits.

2.4.4 Improving Circuit Construction

Circuit Construction in Tor. In the original onion routing design, to construct a circuit, a user creates an *onion* where each layer contains symmetric keys for the corresponding relay on the circuit, and information about the next relay on the path. One problem with this approach is that it did not provide a *forward secrecy* property, meaning that if a router is compromised, it can reveal information about past user communication. This can be done if an adversary records all communication and later obtains a private key of a router and uses it to decrypt the session key messages, thereby obtaining the key to decrypt the rest of the communication.

To avoid this problem, the circuit construction in Tor is performed in a *telescoping* manner. This means that the circuit is built incrementally and interactively with each hop on the path, as described in section 2.3.1. The drawback of this operation is that it is costly; establishing a circuit of length ℓ requires $\Theta(\ell^2)$ network communications, and $\Theta(\ell^2)$ symmetric encryptions/decryptions [KZG07].

Øverlier and Syverson [ØS07] introduced four protocols that aim to reduce the communication and computation overhead of circuit construction in Tor. In their first protocol (which is the basis for all their subsequent protocols), no RSA key is used for circuit construction; private RSA keys are only used by routers to sign their information that they send to directory authorities. For circuit construction, every router creates DH parameters $DH_{x,pub/priv}$ and publishes the public values with the rest of its information in the directory servers. Clients create circuits in a similar way to the original Tor method except that a client uses the public DH keys of the selected routers on the circuit to encrypt the session keys instead of the RSA public keys. The second protocol uses the first protocol and creates a circuit by sending one command cell to build a circuit in one pass. The insight of the third protocol is that since the link between the client and the first router is TLS encrypted, there is no need to use a DH key exchange, but they can simply exchange symmetric keys. Finally, the fourth protocol proposes a new key agreement protocol using both the ephemeral keys of both the client and router and the long-term keys of the router; however, this fourth protocol has been broken [GSU11].

Pairing-Based Onion Routing [KZG07]. In this work, Kate *et al.* propose replacing the circuit construction scheme in Tor with a pairing-based onion routing (PB-OR) protocol that uses a pairing-based non-interactive key agreement protocol. In order for their scheme to achieve unilateral anonymity (meaning that the client authenticates a relay without leaking the client's identity), they use an identity-based infrastructure. A trusted entity known as the *private key generator* (PKG) takes a router's well-known identity ID, and uses a master key only known to the PKG to generate a private key d for the router. The client uses the ID to generate as many pseudonyms as it needs. Then, the client achieves anonymity during establishing the key agreement phase with routers by presenting a different pseudonym with each router; routers use their private keys d to complete the key agreement. Because the key agreement protocol is non-interactive, it significantly reduces the communication overhead of the circuit construction compared to Tor, and it allows a circuit to be constructed in one pass. However, the PKG is able to decrypt all messages encrypted for clients, a single-point-of-failure-problem. Also, to maintain forward secrecy, routers are required perform costly communications with the PKG in order to change their identity keys frequently.

Certificateless Onion Routing [CFG09]. In this work, the authors note the problems inherited from the use of a PKG in the above scheme and propose to improve it by replacing the anonymous pairing-based key agreement with an anonymous certificateless key agreement scheme. The idea

of this scheme is that a client obtains *partial secret keys* from the trusted entity (key generation center KGC), from which he can compute public/secret key pairs PK and SK. Therefore, the newly computed private key SK is not known even to the KGC, and the pair PK/SK can be used to generate several pseudonyms as needed. The rest of the protocol is very similar to that of PB-OR. Also, another advantage with this approach is that routers can update their keys locally without contacting the KGC.

2.4.5 Attacks on Tor

The traditional analysis of the security of Tor states that if an adversary controls a fraction f of the network, then the probability of circuit compromise is f^2, which is the probability of controlling the two ends of a circuit. However, there have been several effective attacks on Tor, and they can be categorized into three types: passive attacks, path selection attacks, and side channel information attacks.

Passive Attacks

AS-Level Adversary. Edman and Syverson [ES09] argue that the security guarantees provided by Tor are not as originally thought, especially in the face of an AS-level adversary. An AS is an independent network under the control of an operator, and the Internet consists of several interconnected ASes. If the same AS appears on the path between the client and its entry guard and also appears on the path between the exit and the destination, the AS can use traffic analysis to de-anonymize the user. To understand the threats of an AS-level adversary, the authors used available routing information databases (RIBs) in order to construct AS-level graphs that depict ASes and their relationships and adjacencies using path inference algorithms. Then, they used a shortest path algorithm to compute paths between the ASes. Experimental results have shown that the probability that a single AS appears at the two ends of a circuit can be as high as 20%. This probability can be slightly decreased using an AS-aware or country-aware relay path selection algorithm. It is worth noting that the modifications in the router selection algorithm of Tor, such as enforcing a different /16 subnet for routers on a path, or the weighted bandwidth selection of routers, have had improvements in limiting the threat of an AS-level adversary.

Path Selection Attacks

Selective Denial of Service. Although this attack [BDMT07] was considered in Tor's threat model, its impact was not analyzed. The selective denial of service attack works by disrupting

the reliability of the system with the goal of reducing its security. In this attack, the attacker simply denies service to circuits that he cannot compromise (by appearing at the two ends of a circuit). For example, if an entry guard is malicious, it will not allow its client to have a reliable anonymous communication except if the exit relay is also a colluding node (an entry guard can determine if the exit node is a colluding node using traffic analysis). A circuit is reliable if all relays are reliable and either all relays are honest, or at least two edge relays are compromised. Assuming all nodes are highly reliable in the system, the security of the system can be as low as 50% even when the fraction of honest nodes is as high as 80%.

Low-Resource Routing Attack. This attack [BMG+07] works by exploiting the path selection algorithm of Tor and influencing it to select the malicious relays. The adversary either installs high-bandwidth nodes, or even low-resource nodes that advertise high-bandwidth capabilities. When clients establish circuits, they will be trapped into selecting the malicious nodes with a higher probability because the router selection algorithm of Tor biases its selection towards higher-bandwidth relays. To increase the effectiveness of the attack, malicious nodes perform a selective disruption where they refuse to relay traffic unless they are able to control the entry guard and the exit node of a circuit. The authors also describe an attack that enables the malicious routers to confirm that they are controlling the entry and exit positions of a circuit. To perform this attack, each malicious router logs some statistics and information regarding its connection, such as the IP addresses and ports and some connection timestamps, and reports the logs to a colluding centralized authority which runs the circuit linking analysis in real time. Experiments on an isolated Tor network have revealed the success of this attack. For example, if an attacker controls 10% of the network, it can compromise as many as 47% of the constructed paths.

Side Channel Information Attacks

Throughput Fingerprinting. One of the problems facing Tor is the heterogeneity of its resources. This problem manifests itself as users build circuits that have distinctive characteristics that can be enough to fingerprint them. Mittal *et al.* [MKJ+11] present a number of throughput fingerprinting attacks. The insight of their attacks is that if two circuits share the same path or even just the bottleneck node on the path, their throughput observations would be highly correlated. This allows an adversary to passively identify if two circuits share a sub-path or just a bottleneck. Also, an attacker can confirm if a specific relay R carries a flow f by probing R and computing the throughput correlation between f and R. Furthermore, two separate malicious servers can confirm if two streams belong to the same circuit (user). These attacks yield accurate results; however, they are costly. The cost of the attack scales as Θ(N) for the probing operations, which must be performed throughout the duration of the communication, where N is the number of possible relays.

Congestion Attacks. The first low-cost congestion attack on Tor was described by Murdoch and Danezis [MD05]. The main contribution of this work is the realization that a global adversary is not necessary to perform traffic analysis attacks on Tor. The adversary, which can be a malicious server interested in learning the identities of its clients, is required to install a relay in the network. When a client connects to the malicious server, the server responds to the client with data modulated in a very specific traffic pattern. The adversary then can learn the relays on the path by performing probing tests on the suspected relays. Experimental results on live Tor nodes showed that when a relay in the client's circuit is monitored, that relay exhibited a probe latency that was highly correlated with the modulated traffic. The authors witnessed a very high success rate with few false positives. This attack was carried out in 2005, when Tor consisted only of 50 nodes, and was not as heavily used as today.

Another congestion attack was introduced by Evans *et al.* [EDG09]. The goal of this attack is simply to identify the entry guard of a client. The attack is carried out as follows: a client connects to a server using a malicious exit relay, which injects JavaScript code into the client's requested web page. Next, since many Tor users do not disable JavaScript, the script would generate a signal with a specific pattern through the client's circuit, and thereby keep it alive. The attacker monitors the arrivals of the requests at the server, and records a baseline latency. The attacker then constructs a long circuit (preferably high bandwidth) that passes through the target suspected relay many times. From the target relay's point of view, this long path is multiple different circuits. Then, the long path circuit is used to congest the target relay and if the target relay is indeed the client's entry guard, the malicious server will observe a correlation between the latency of the signal and the duration of the congestion attack.

Network Latency. In the examples of side-channel information attacks described above, the goal of the adversary is deanonymize the Tor routers in a circuit, or to compromise the unlinkability of streams. Hopper *et al.* [HVCT07] present two attacks that aim to reduce the anonymity of clients. In the first attack, known as the circuit linkability attack, two malicious servers seek to find out if two different connections coming from one exit relay belong to the same circuit (client) or not. The two streams are assumed to belong to two separate circuits and the distribution of the latency of each circuit is measured (from the common exit relay to the client). If both circuits have the same distribution, then the two streams should appear to come from the same circuit. The second attack aims to approximate a client's location using a combination of the Murdoch-Danezis low-cost congestion attack and a latency attack. First, when a client is communicating through the Tor network, the congestion attack is carried out in order to de-anonymize the relays used in the circuit, and in particular to identify the entry guard of the client. The adversary's next goal is to measure the latency between the victim's client and its entry guard. This can be estimated by using a colluding Tor client to construct an identical circuit and measure the latency of the circuit to infer the latency of the link in question. Both attacks presented in this work have been

tested and they both are successful in reducing the entropy of the anonymity set distribution. The authors suggest that to mitigate such attacks, it may be necessary to introduce artificial delays or use an enhanced path selection algorithm.

Summary. In this section, we presented several attacks that have shown their effectiveness in undermining the anonymity of the Tor network and its users. Since this thesis is concerned with improving the performance of Tor without weakening its threat model, we ensure that our proposed improvements, which we present in the following chapters, do not introduce new attacks or strengthen the effectiveness of the existing attacks described.

Chapter 3

The Path Less Travelled: Overcoming Tor's Bottlenecks with Multipaths

Recall that Tor is the most popular low-latency anonymity network for enhancing ordinary users' online privacy and resisting censorship. While it has grown in popularity, with a volunteer-operated infrastructure of Tor routers serving hundreds of thousands of daily users, Tor has a variety of performance problems that result in poor quality of service, a strong disincentive to use the system, and weaker anonymity properties for all users. We observe that one reason why Tor is slow is due to low-bandwidth volunteer-operated routers. When clients use a low-bandwidth router, their throughput is limited by the capacity of the slowest node.

With the introduction of bridges, low-bandwidth Tor routers are becoming more common and essential to Tor's ability to resist censorship. In this chapter, we present Conflux, a dynamic traffic-splitting approach that assigns traffic to an overlay path based on its measured latency. Because it enhances the load-balancing properties of the network, Conflux considerably increases performance for clients using low-bandwidth bridges. Moreover, Conflux significantly improves the experience of users who watch streaming videos online.

Through live measurements and a whole-network evaluation conducted on a scalable network emulator, we show that our approach offers an improvement of approximately 30% in expected download time for web browsers who use Tor bridges and for streaming application users. We also show that Conflux introduces only slight tradeoffs between users' anonymity and performance.

3.1 Introduction

One cause of the degraded performance in Tor is the diversity of bandwidth provided by Tor's volunteer-operated routers, and in particular the low-bandwidth bridges. In this chapter, we recognize the significance of improving the experience of clients that use bandwidth-limited bridges. We also recognize the need to enhance the performance of some high-throughput applications, such as streaming web videos, for Tor users. We propose an unconventional approach to improving performance when using low-bandwidth routers and bridges: *Tor users should split their traffic across multiple semi-disjoint circuits.*

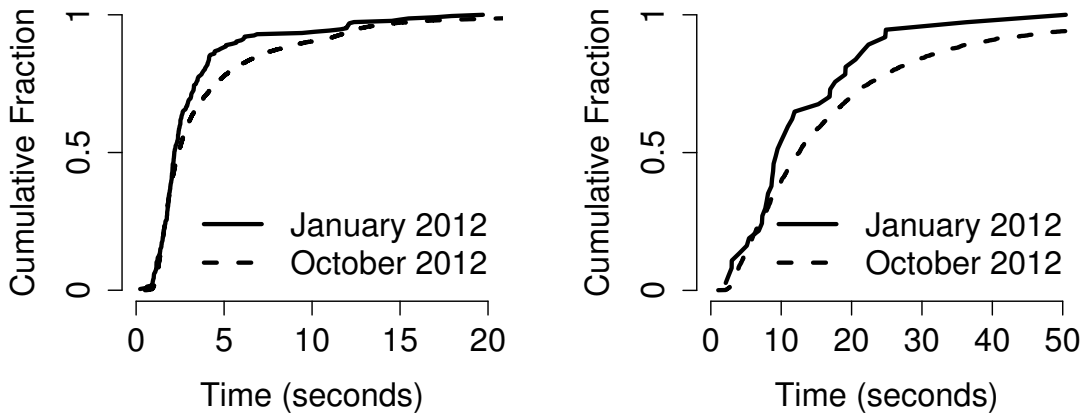
Dynamic Traffic Splitting for Tor. Recall that Tor users construct circuits periodically and use each for approximately ten minutes. Each user's traffic is divided into fixed-size (512-byte) cells, and these cells are transported along the circuits. In the context of Tor, traffic splitting offers the following benefits:

- *Improve load balancing.* When routers become over-utilized and experience congestion, splitting traffic across semi-disjoint paths can ease the burden on the congested circuit; under our scheme, circuits need only share a common exit router.
- *Improve performance with low-bandwidth relays.* By splitting data over multiple circuits, the user's throughput can achieve up to the aggregate throughput of all circuits rather than a single one. This is particularly useful when a circuit uses a low-bandwidth router. Tor's router selection algorithm favors routers that have higher bandwidths to ensure sufficient throughput to transport users' traffic and to balance the traffic load across Tor's routers. However, individual Tor routers can have vastly different bandwidth capacities, ranging from 20 KiB/s to over 20 MiB/s. Figure 3.1 shows a long-tailed distribution of download times for 50 KiB and 1 MiB files over the course of two different months: January and October 2012.[1] These slower downloads often correspond to circuits that used at least one low-bandwidth router. By combining multiple circuits with low-bandwidth nodes, the attainable throughput is no longer bound by the bottleneck node, but is instead the aggregate of each individual circuit's throughput.

Our approach. We design, implement, and evaluate *Conflux*,[2] a novel congestion-aware traffic splitting and load balancing algorithm for anonymous communication networks. Conflux forwards a client's individual cells down multiple circuits that share a common exit router. Our algorithm dynamically measures the throughput of each constituent circuit and assigns traffic to each in proportion to its observed throughput. Our approach performs sub-stream traffic splitting, which provides a fine granularity of load balancing, as splitting can be performed at the

[1] This data was obtained from The Tor Metrics Portal [To12b].

[2] Conflux: a flowing together of rivers or streams.



(a) 50 KiB (b) 1 MiB

Figure 3.1: Time required to download files over Tor in January and October 2012. Observe the long tails in each case.

individual cell level. This allows the traffic that is sent on a circuit to correspond to its desired load. The circuit's endpoints (the client and the exit router) are responsible for splitting the traffic at one endpoint and buffering, re-ordering, and delivering in-order cells to the application at the other end of the circuit. This approach can be deployed incrementally, as only clients and exit routers need to upgrade to support it.

To quantify the performance benefits of our proposed design, we perform a variety of live and whole-network experiments on an emulation-based Tor network testbed [BSMG11]. Our evaluation indicates that utilizing multiple circuits with Conflux can result in decreased queuing delays and increased throughput for users, particularly those who rely on low-bandwidth bridges to access the Tor network. We also find that, under light traffic loads, Conflux improves performance for clients who use Tor to access streaming videos (such as blocked YouTube videos[3]). Improving performance for such users is important, as streaming video websites are becoming a dominant source of Internet traffic [MFPA09, Sa11].

We also critically evaluate the security implications of utilizing additional circuits in light of the well-studied end-to-end traffic confirmation attack [SS03, SW06]. Our analyses indicate

[3]Note that while Tor's browser bundle disables Flash by default, it is now possible to stream videos over Tor using HTML5. We expect this use case of streaming video over Tor to increase in popularity in the near term.

that our scheme only slightly increases the users' vulnerability to this attack. Anonymity is also slightly decreased when the adversary uses powerful selective denial of service tactics [BMG+07, BDMT07, THK09, DB13] to maximize the number of circuits that can be compromised.

Contributions. This chapter offers these contributions.

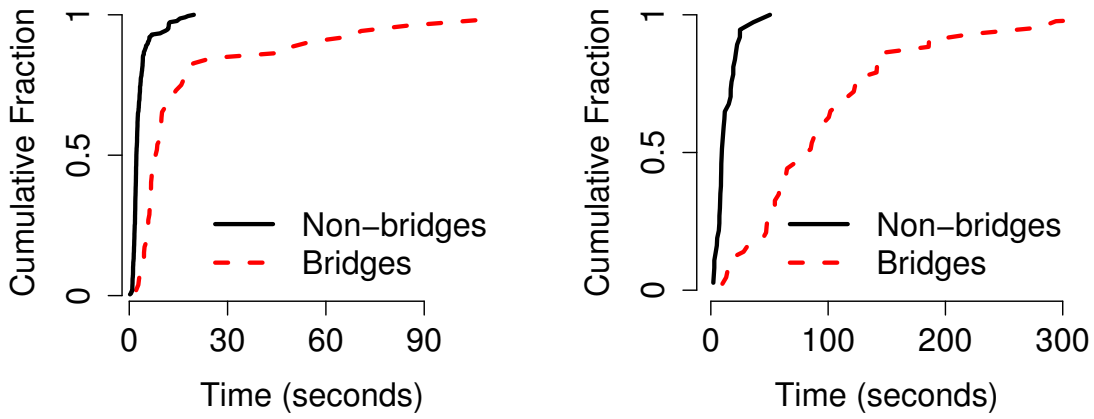
- We motivate for the crucial need to improve the Tor experience for bridge and streaming application users. To our knowledge, this is the first work that sheds light on the performance problems for those two emerging classes of users.
- To improve performance for bridge and streaming application users, we design, implement, and evaluate a dynamic traffic splitting scheme that distributes the traffic load across circuits according to each circuit's bandwidth capacity.
- Our live performance analysis indicates that Conflux results in an expected improvement of 30% in a typical Tor client's queuing delay and up to 75% in total download time. Whole-network experiments show that noticeable improvements are possible even when most or all clients adopt Conflux.
- We analyze the security of Conflux and provide quantitative results showing that there is a small tradeoff between users' anonymity and performance gains.

Outline. The remainder of this chapter is organized as follows: Section 3.2 motivates for the need of multipath routing for Tor. Section 3.3 presents the design of Conflux and an algorithm for splitting traffic in a manner that balances the traffic load over each circuit. We evaluate our proposal in Section 3.4 and offer a security analysis in Section 3.5. We discuss a variety of open issues and enumerate avenues of future work with our design in Section 3.6. We contrast our contributions with related work in Section 3.7 conclude in Section 3.8.

3.2 Motivation

3.2.1 Evading censorship with bridges

In addition to anonymous communications, Tor is an important tool in the fight against censorship. Tor helps users around the world visit blocked websites. In some cases, Tor's infrastructure of directory authorities and routers has been blocked, for example by the so-called "Great Firewall of China" [Lew10]. To facilitate entry to the Tor network despite such blocking, Tor has introduced *bridges*, which are unlisted Tor routers that are distributed to censored users via out-of-band mechanisms such as HTTPS queries to `bridges.torproject.org`. To ensure that a censor cannot collect all bridges and simply block them, Tor currently limits the number of



(a) 50 KiB (b) 1 MiB

Figure 3.2: Download time comparison between Tor users who use public entry guards and those who use bridges. Note that bridge downloads are considerably slower, particularly in the tails.

bridges that are distributed to each /24 IP address block [To09]. Currently, clients use bridges in lieu of an entry guard, to keep the total circuit length at three routers.

While bridges provide an essential service to an estimated 30,000 censored users as of November 2012 [To12d], they are believed to be operated by Tor clients who often reside on low-bandwidth broadband networks. To confirm this hypothesis, we obtained 221, of approximately 700 [To12c], bridges in January 2012 using Tor's standard HTTPS request service from PlanetLab hosts on 55 different /24 IP networks.[4] Figure 3.2 compares the performance of a live Tor client that downloads 50 KiB and 1 MiB files using entry guards versus bridges. Clearly, the low-bandwidth bridges are a significant source of poor performance. Furthermore, Tor bridges are becoming integrated into ubiquitous devices such as wireless access points to simplify the process of configuring and running a bridge on a broadband Internet link at home [To13]. Thus, because low-bandwidth bridges will likely become even more common in the near future, in this work we seek to improve performance for bandwidth-limited bridge clients.

[4]This procedure for enumerating bridges is described in more detail by Ling *et al.* [LLY+12]. Automatically enumerating Tor bridges is more difficult at present because clients have to solve CAPTCHAs. For our experimental evaluation, we use a smaller more recent list of bridges

3.2.2 Adapting Tor to the changing web.

Unlike previous efforts which seek to enhance the experience of web browsers in Tor by throttling bulk downloads (specifically file-sharing applications) [MWS11, JSH12], we recognize that some emerging classes of bulk transfers should actually be improved rather than throttled. In fact, recent Internet traffic studies have revealed that file sharing applications are consuming less bandwidth,[5] while streaming video applications are starting to account for an increasingly large fraction of Internet traffic by volume [MFPA09, Sa11]. Therefore, in order to survive and continue to attract new users, it is crucial for Tor to meet the demands of its users and the changing web by improving the experience for streaming users. Although The Tor Project mainly welcomes web browsing, it is hard these days to separate streaming from web browsing. For example, if a user visits a blocked news website via Tor, the user may also want to view videos associated with the stories accessed.

3.3 Conflux's Design

We next shift attention to the design of our system. An OP that uses Conflux builds a number of circuits (two or more) that intersect at a common exit OR. We refer to the OP and common exit OR as the *end points* of a multipath. The OP receives and sends data to the client's application (such as a web browser), while the exit OR sends and receives data from an external server (such as a web server). Each end point receives data and splits it into cells, adding sequence numbers to the cell headers. Next, the end point divides the cells across the circuits of the multipath according to a traffic splitting scheme. When the other end point receives the cells, it collects and reorders them according to their sequence numbers before delivering their contents to their destinations. We note that this approach does not replace Tor's bandwidth-weighted router selection algorithm, but complements it.

Our approach to cell-level traffic splitting consists of three parts: 1) multipath construction, 2) throughput-informed sub-stream traffic splitting, and 3) sequencing, buffering, and reordering. We next describe each part in turn.

Constructing the multipath. As shown in Figure 3.3, the client constructs the first circuit, called the *primary circuit*, according to Tor's bandwidth-weighted router selection. Then, if the client wishes to use Conflux, the client forms another circuit that uses different entry and middle ORs, which are also selected according to the bandwidth-weighted algorithm. The only constraint our system requires on the second circuit is that its exit OR has to be the same as

[5] Note that P2P traffic on Tor is also likely to drop with the rise of UDP-based P2P applications [BMC+11].

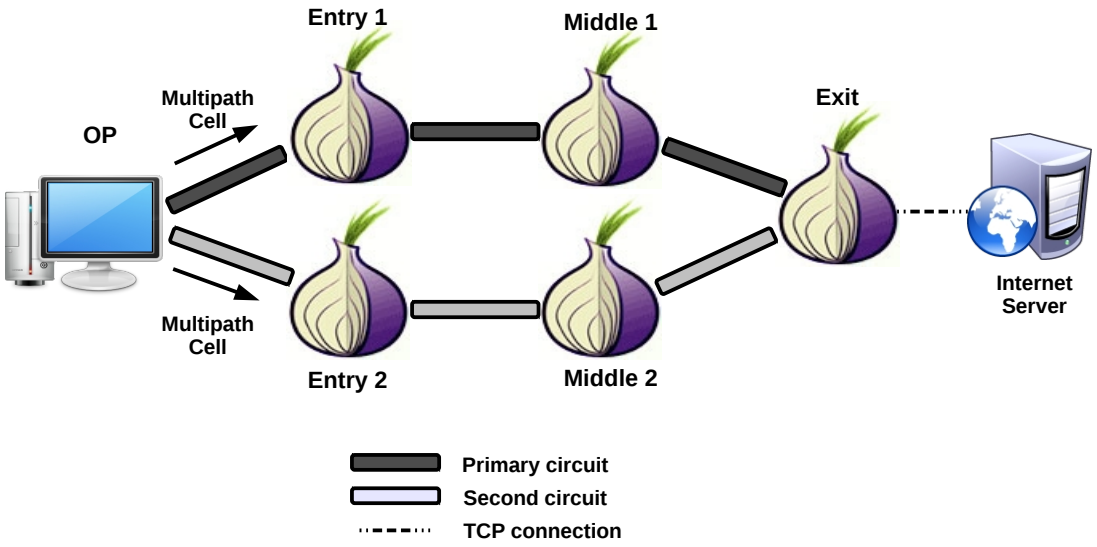


Figure 3.3: Multipath construction and stream linking

the primary circuit's exit OR. Next, the OP sends a new type of command cell, which we call the "multipath" cell, through both circuits to the common exit OR. The multipath cell contains a 32-byte random nonce that is common to each of the OP's linked circuits. This nonce enables the exit router to associate the OP's TCP streams with its linked circuits.[6] The OP uses the primary circuit to command the exit OR to establish the TCP connections to Internet destinations. Closing a multipath is no different from closing circuits in Tor. If a circuit in a multipath exceeds its lifetime (ten minutes by default), and if it is idle, the circuit is torn down. Closing one circuit does not affect the operation of other circuits. Since a Tor client already builds many spare circuits by default, we do not expect any additional load being introduced by Conflux.

Dynamic sub-stream traffic splitting. One approach to performing sub-stream traffic splitting is to perform traffic splitting in a round-robin fashion: cells in a stream are sent one (or a small fixed-sized batch) at a time down each circuit in turn. However, because different circuits have different throughput capacities, sending equal amounts of traffic down each circuit may not result in optimal load balancing. To solve that problem, we designed and implemented a dynamic load balancing algorithm where the splitting end point assigns different amounts of traffic to each

[6] As usual, a malicious exit router can trivially observe all circuits it handles, including linked ones.

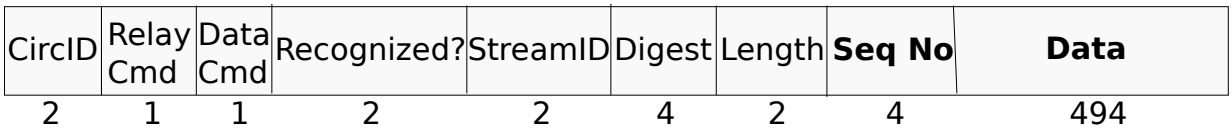


Figure 3.4: The 512-byte data cell format with each field's length (in bytes) for Conflux.

circuit depending on its observed throughput relative to the other linked circuits. The advantage of this approach is that it is reactive to network dynamics, such as the congestion state of a circuit and its available capacity relative to the other circuits.

This scheme works as follows. First, the splitting end point (the OP for client-to-server traffic, or the exit OR for server-to-client traffic) measures the latency of each of the linked circuits. This can be done by storing the time every 100^{th} cell is sent down a particular circuit, and noting the time that the corresponding circuit-level `SENDME` arrives. (Recall from Section 2.3.3 how the circuit-level `SENDME` works.) This allows the splitting end point to compute the current round-trip-time (RTT) of cells on the circuit; this will be inversely proportional to the circuit throughput, as cells are of fixed size.

The splitting end point periodically updates the throughput measurements assigned to each linked circuit. Next, every time a data cell is ready to be transmitted on a multipath, the particular circuit used to send the cell is selected with a probability proportional to its throughput.

Sequencing, buffering, and reordering. Before any splitting can be performed on TCP streams across different circuits, we have to ensure that the receiver will be able to reorder cells before delivering them to their destination (client program or exit TCP connection). Therefore, we implement sequencing of data cells before sending them down our multipaths. Tor's standard data cell consists of a circuit identifier, a "relay" command type, a "data" sub-type, a "recognized" field to identify whether the cell is to be delivered locally, a stream identifier, a message digest to ensure integrity, a data length, and the data. We modify the cell format slightly, to reserve the first four bytes of the payload for the sequence numbers, as shown in Figure 3.4. This reduces the amount of data that can fit into each cell's payload by less than 1%.

Because we divide a single TCP stream across circuits, we expect that cells may arrive out of order. Therefore, the two end points of a multipath, the OP and exit OR, are responsible for buffering and reordering cells that arrive out of order. First, as long as the cells arrive in order, they are immediately delivered to the client application (or the TCP exit connection) when the receiver is the OP (or the exit OR). Also, we keep track of the sequence number of the last delivered cell. If a cell arrives out of order, it is stored in a sorted list of cells. When the next

expected cell arrives, it is delivered to the OP (or TCP exit connection) along with any buffered cells with subsequent sequence numbers that have already arrived.

Implementation details. We implemented the multipath construction and cell sequencing, buffering, and reordering in the Tor source code (version 0.2.3.0-alpha-dev). We also implemented the weighted traffic splitting algorithm as described above. Conflux can be turned on or off as a configuration option. Note that only the circuit's end points (e.g., the client and the exit router) are required to upgrade to run Conflux. Thus, Conflux can be incrementally deployed as individual exit routers and clients upgrade. Our full implementation consists of roughly 2,000 lines of code.

3.4 Performance Evaluation

In order to empirically demonstrate the potential performance benefits of the proposed scheme, we present a series of experiments. In particular, we wish to understand the potential benefits of Conflux on the currently deployed live Tor network. We also conduct experiments in an isolated, network emulation environment to explore how Conflux might perform at scale, when adopted by many or all Tor clients and routers.

3.4.1 Experimental Setup

Live experiments. First, we seek to measure the potential benefits of deploying a modified Tor router on the currently deployed Tor network. Since only the Tor exit router needs to be modified in order to use Conflux, we deploy a single exit router and conduct a series of performance measurements using a Tor client that we control.

Each measurement is collected as follows: First, a Conflux circuit is built using two entry routers $entry_1$ and $entry_2$, two middle routers $middle_1$ and $middle_2$, and our modified exit router *exit*. In order not to expose other clients' traffic, we set the exit policy of *exit* so that it can only connect to a specific web server. This means that *exit* will act as an exit router only for our traffic, but it can be a middle or an entry node for other clients' encrypted traffic. Using Conflux, our client fetches 320 KiB, 1 MiB and 5 MiB files. These file sizes were chosen to approximate web pages and larger files [Ram12]. For comparison, the stock Tor client downloads the files over different circuits it builds using Tor's default bandwidth-weighted router selection algorithm.[7]

[7] To reduce any bias in the performance results due to the selection of particularly fast or slow entry guards, we disable the use of entry guards for this experiment.

Table 3.1: Network model for whole-network experiments

Attribute	*Data Source*
Pairwise link latency	King dataset [GKL+09]
Tor router bandwidth	Tor consensus (Nov. 2011)
Tor client bandwidth	Ookla Net Index dataset [Oo]
Traffic characteristics	Tor traffic study [MBG+08]

Measurements were collected from December 2012–January 2013, during which time our exit router was configured with a bandwidth rate of 200 KB/s.

To evaluate the performance benefits for people using low-bandwidth Tor bridges, we also collect measurements where our client uses Tor bridges as its entry nodes. We collected 36 bridges by using Tor's standard HTTPS request service and we manually solved CAPTCHAs. The bridge clients work as follows. Every thirty minutes, our stock Tor and Conflux clients choose six bridges randomly from the list of 36 bridges we obtained, and use them as the first hop on each circuit they construct.

Whole-network experiments. One of the limitations of a live performance evaluation is that it is generally not possible to understand how performance might change when all participants of the network adopt the new design. To help understand these *whole-network effects*, we also perform experiments using the ExperimenTor testbed [BSMG11]. ExperimenTor is a Tor network testbed and toolkit that enables whole-network Tor experiments on a network topology with realistic delay, bandwidth, and other characteristics using the Modelnet [VYW+02] network emulation platform.

Briefly, our Modelnet setup consists of two machines, an emulator node and a virtual node. The virtual node runs the Tor network, which consists of directory authorities, ORs and OPs. The virtual node also runs the destination servers. Communication among the different nodes on the Tor network and the destination servers is routed through the emulation node, which provides the underlying IP network emulation. Several network parameters such as the bandwidth, propagation delay and drop rate can by configured on the network topology deployed on the emulator node to provide a realistic underlying network emulation.

One challenge in conducting such an evaluation is that one must faithfully model the important dynamics of the live network such as network latency, bandwidth, and traffic characteristics and replicate them in isolation. In an effort to enhance the realism of our experiments, we use a variety of empirical data sources, summarized in Table 3.1, to construct a network topology based on realistic link latencies, Tor router bandwidths sampled uniformly from a Tor consensus

document from November 1, 2011, and asymmetric Tor client bandwidths assigned by sampling from the Ookla Net Index broadband data set [Oo] (the interquartile ranges are 4–13 Mbit/s downstream, 0.5–1.9 Mbit/s upstream).

Client traffic models. In addition to building a realistic network topology, it is important to replicate the dynamics of the network's traffic. Since Tor's users are anonymous, it is inherently difficult to characterize real Tor traffic. One such study [MBG+08] exists, which reported that over 92% of TCP connections leaving a Tor exit router result from web browsing and make up nearly 60% of the network's aggregate traffic volume. However, BitTorrent accounts for only about 3% of connections, but comprises over 40% of the aggregate traffic volume. We employ these empirical observations in developing realistic traffic models for our whole-network experiments. Beyond modeling the dynamics of the past and present Tor network, we also consider emerging trends in Internet traffic.

We model two types of clients in our experiments: First, web browsing clients are modelled as fetching 320 KiB files (the median web page size on the Internet [Ram12]) with random think time pauses between 1–30 seconds (chosen uniformly at random). A similar distribution of "think times" between web requests was measured by Hernández-Campos *et al.* [HCJS03]. Second, bulk clients (e.g., streaming video) download 5 MiB files with uniformly random delays of 1–5 minutes between fetches. This download size and delay distribution approximates observations of YouTube video sizes and viewing durations [Kin12].[8]

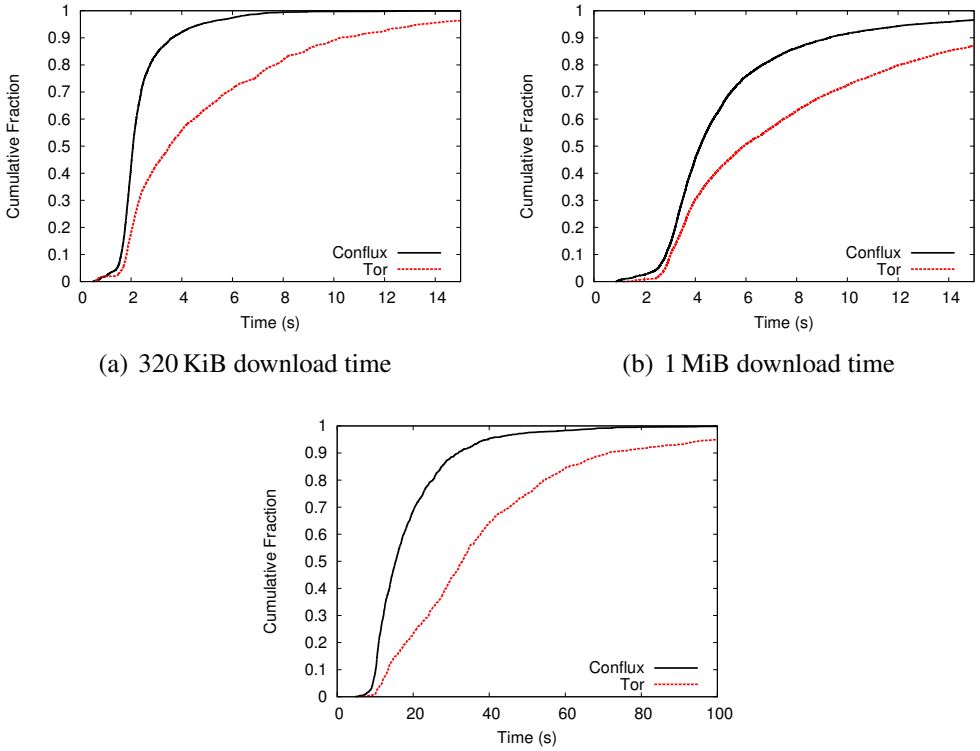
To highlight the performance benefits for bandwidth-deprived bridge clients, we also conduct whole-network experiments in which clients use a bridge in lieu of an entry guard. Since bridges are typically run by Tor clients themselves, we configure five bridges to run on asymmetric broadband-like Internet connections chosen from the Ookla Net Index data set.

3.4.2 Results

Performance metrics. To evaluate the performance of our technique, we measure *time-to-first-byte* and *download time*. The time-to-first-byte is the time it takes the client to receive the first cell of data after it issued a request; it is a good measure of both the responsiveness of the network and its congestion state. The time-to-first-byte is two end-to-end circuit RTTs: one RTT to connect to the destination web server, and a second RTT to issue a request for data (*e.g.*, HTTP `GET`) and receive the first byte of data in response.[6] Improving the time-to-first-byte is especially important

[8] Note that streaming websites such as Youtube and Netflix use different strategies for streaming, one of which can be considered a simple file download [RLL+11]. We chose this strategy in our experiments to simplify the experimental setup, as we run our experiments in an isolated testbed.

[6] Note that there is a proposal being considered to eliminate one of these RTTs [Gol10, Gol11].



(a) 320 KiB download time

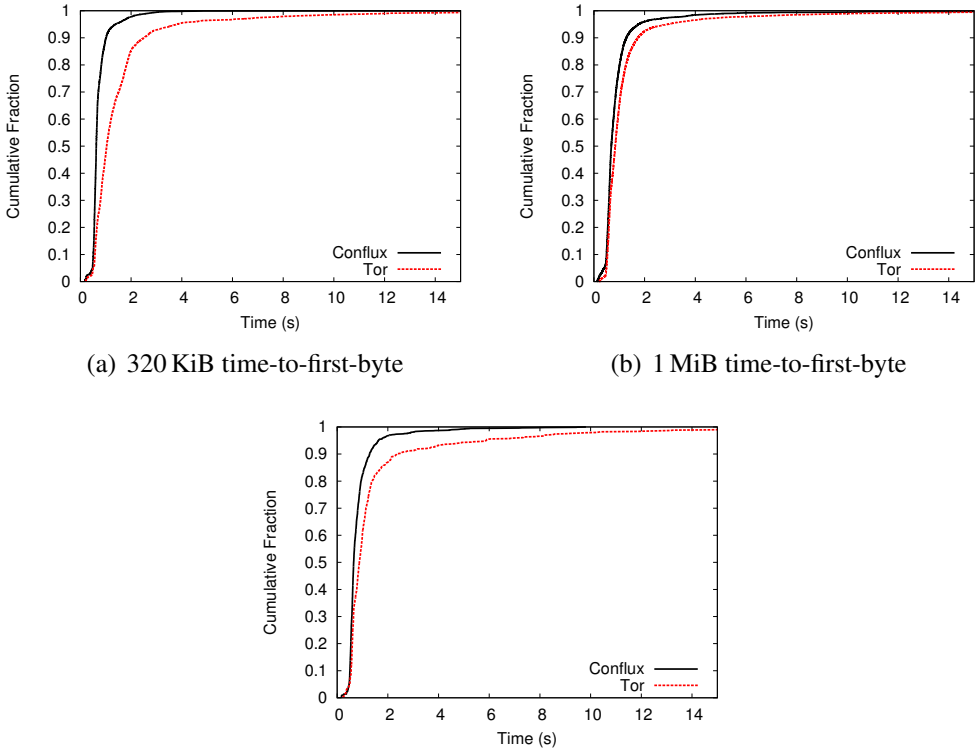
(b) 1 MiB download time

(c) 5 MiB download time

Figure 3.5: Download time live performance comparison between Tor and Conflux

for interactive applications such as web browsing, as it directly informs the time between a user click and when the screen starts to change. In fact, prior work found that interactive users (such as web browsing users) have a low tolerance to delays beyond a few seconds [Nah04]; thus, ensuring fast responses for interactive traffic is essential to Tor's usability. The other metric we consider is download time, which is simply the time it takes for the client to receive the last byte of data after issuing a request (download time includes the time-to-first-byte). We consistently use the time-to-first-byte and download time throughout the remainder of this thesis to evaluate the effectiveness and performance benefits of our proposed improvements.

Live performance. Figure 3.5 compares the time for a client to download 320 KiB, 1 MiB, and 5 MiB files using Tor and Conflux. We observe a noticeable improvement in download times for all file sizes. However, improvements are more visible with larger file sizes. For example, the median improvement in download time for the 320 KiB and 1 MiB files is 42% and 33%,



(a) 320 KiB time-to-first-byte

(b) 1 MiB time-to-first-byte

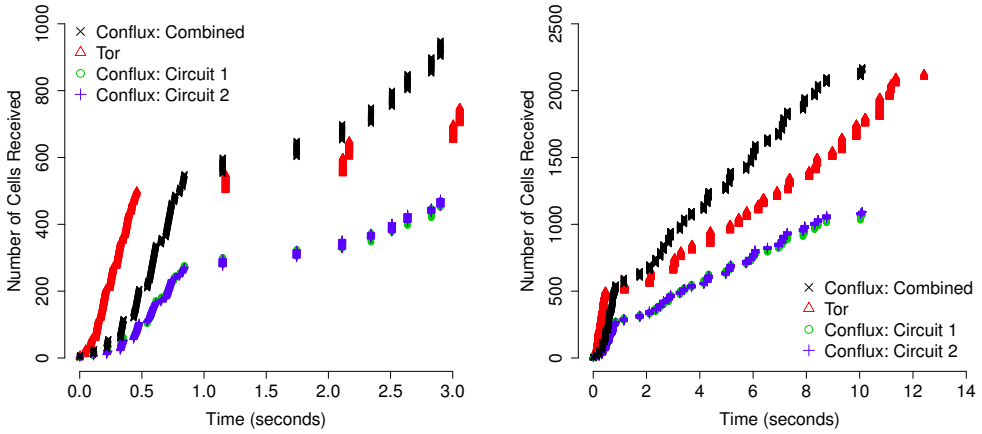
(c) 5 MiB time-to-first-byte

Figure 3.6: Time-to-first-byte live performance comparison between Tor and Conflux

respectively. For the 5 MiB files, the improvement is around 54%. Figure 3.6 also shows the time-to-first-byte comparison between Tor and Conflux for the different file sizes.

The reason why improvements are more visible for larger file sizes is the mechanics of TCP congestion control. When a file download starts, *slow start* initially sets the TCP congestion window (cwnd) of the connection between the exit router and destination server to a small value (typically one segment size). The congestion window exponentially increases by a factor of two every RTT (round trip time) until reaching the slow start threshold (ssthresh). At this point, the linear growth *congestion avoidance* phase starts [APB09].

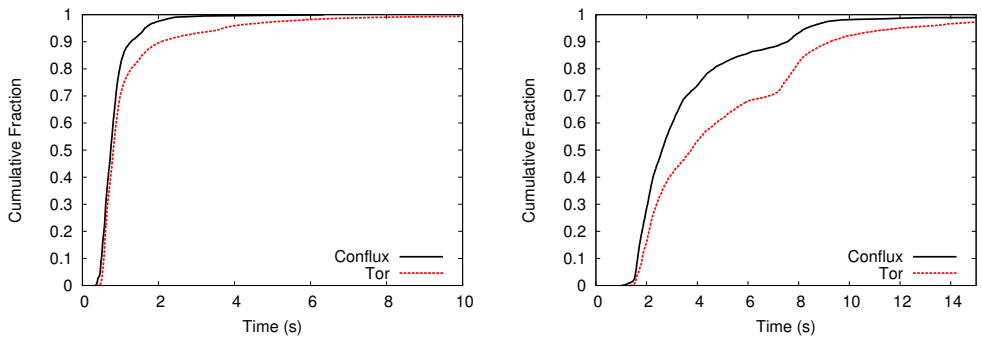
The exponential growth during slow start and the linear growth during congestion avoidance can be seen in Figure 3.7(a). During slow start, the link between the exit router and the destination server is the slowest (bottleneck) link on the Conflux circuits and, consequently, we see no improvement in download time for clients whose transfers complete prior to the congestion avoidance phase. However, for larger downloads, Figure 3.7(b) shows that during congestion



(a) Download rate comparisons during the TCP slow start phase

(b) Download rate comparisons for a 1 MiB download

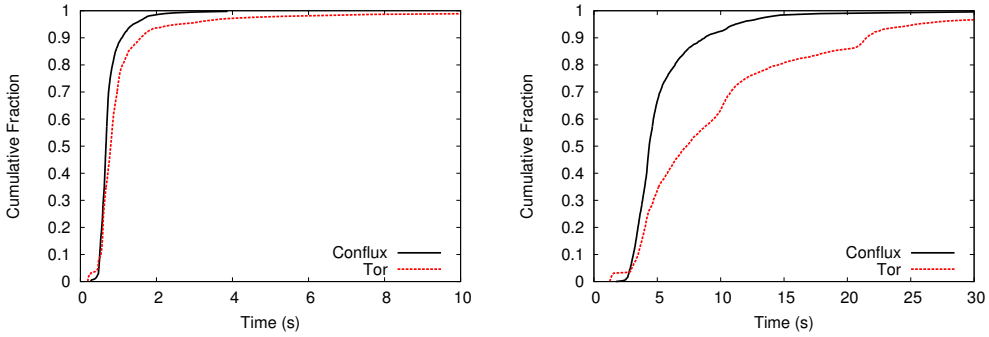
Figure 3.7: Typical download rates for Conflux circuits individually, together and in comparison to a Tor circuit for a 1 MiB file download, as measured by an exit router. (a) indicates that the performance of smaller downloads that complete during TCP slow start experience little noticeable difference between Conflux and Tor because the TCP connection between the exit node and the destination server is the bandwidth bottleneck. (b) shows that Conflux is faster than Tor for downloads that are larger than 600 cells (300 KiB).



(a) 320 KiB time-to-first-byte

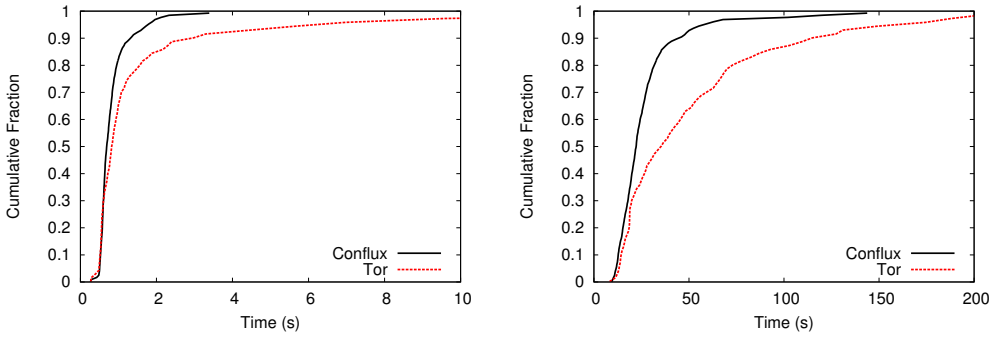
(b) 320 KiB download time

Figure 3.8: Performance comparison for clients that download 320 KiB files between Tor and Conflux using live Tor network bridges



(a) 1 MiB time-to-first-byte (b) 1 MiB download time

Figure 3.9: Performance comparison for clients that download 1 MiB files between Tor and Conflux using live Tor network bridges



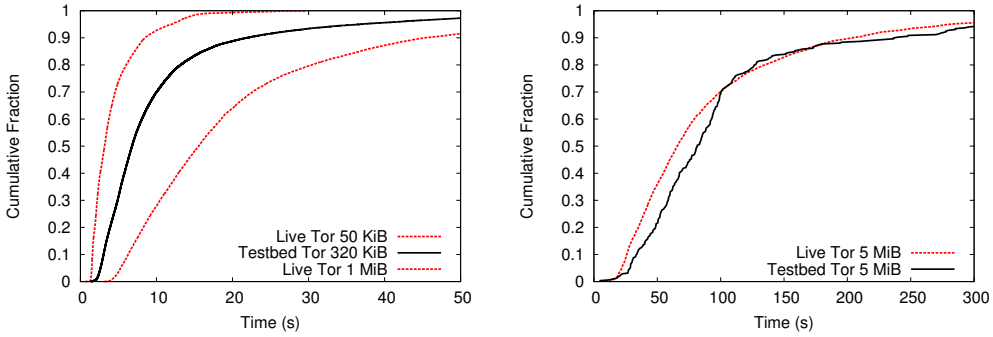
(a) 5 MiB time-to-first-byte (b) 5 MiB download time

Figure 3.10: Performance comparison for clients that download 5 MiB files between Tor and Conflux using live Tor network bridges

avoidance, Conflux offers a higher download rate relative to Tor, which translates to an improved quality of service for such downloads. Thus, we expect Conflux to be most effective at improving performance for users whose downloads do not complete during slow start; this includes the users of applications like streaming video or file sharing.[9][10]

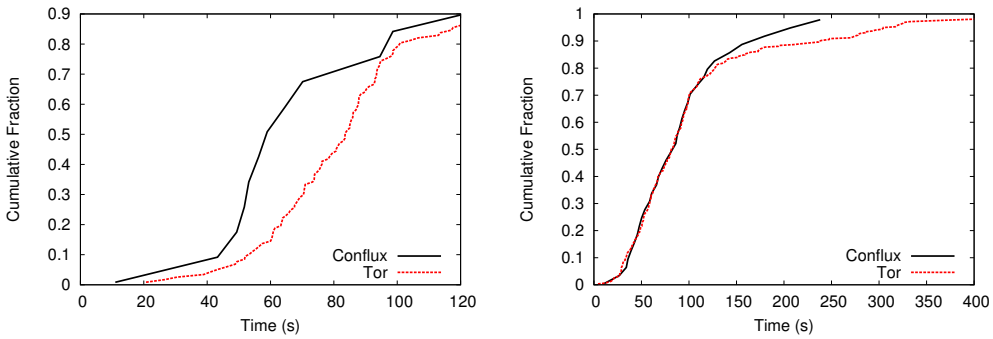
[9]We further confirm our conjecture regarding slow start by running a whole-network experiment where we used persistent TCP connections between exit nodes and servers. Indeed, the web browsing clients in that experiment observed improved performance because they were able to bypass the slow start effects.

[10]We note that this is a well-known performance problem due to the poor interaction between TCP with the short



(a) (b)

Figure 3.11: Comparison between the performance of torperf (Live Tor), and our scaled-down (400 clients and 20 routers) testbed Tor network (ExperimenTor)

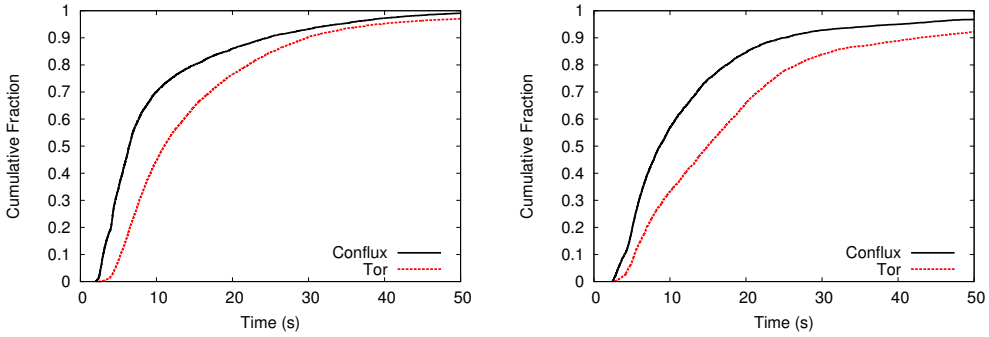


(a) 10 bulk and 390 web browsing clients (b) 30 bulk and 370 web browsing clients

Figure 3.12: Whole-network deployment experiments for the bulk downloaders that download 5 MiB files

Live performance for bridge users. When we apply Conflux with clients who use low-bandwidth bridges as their entry nodes, we also observe significant improvement in performance, regardless of the download size. Figures 3.8(a), 3.9(a) and 3.10(a) compare the time-to-first-byte and the download times for clients who use bridges to download 320 KiB, 1 MiB, and 5 MiB files using Tor and Conflux. Regardless of the download size, the Conflux clients experience faster

and bursty nature of web traffic [All10, CDC11], not an issue inherent to Conflux or Tor. SPDY [Ch] has been proposed to ameliorate this problem.



(a) 10 bulk and 390 web browsing clients (b) 30 bulk and 370 web browsing clients

Figure 3.13: Download time comparison between Tor and Conflux for web clients that download 320 KiB using bridges in whole-network deployment experiments

response times compared to Tor. By using multiple circuits, if one circuit is congested, Conflux is able to send cells down a second, possibly uncongested circuit.

Furthermore, Figures 3.8(b), 3.9(b) and 3.10(b) show that the overall download times for 320 KiB, 1 MiB, and 5 MiB files are significantly improved for the majority of download trials; the performance improvement is most significant for the 5 MiB download, which experiences an improvement of over 50% relative to Tor.

Whole-network deployment. In our previous experiments, we have shown the performance benefits for a single client made possible by using Conflux. In this section, we seek to evaluate the performance of our technique if all our browsing bridge users and bulk clients in the network upgraded. In our large-scale experiments, we deploy a 20-router Tor network on our ExperimenTor testbed. Next, we fix the number of the total Tor clients to 400. Of the 400 clients, 30 clients act as bulk downloaders and 370 clients act as the interactive web browsers. Our initial experiments revealed that if all web clients who use entry guards use Conflux, no performance benefits can be achieved because exit routers would be slower than the entry guards and would become the bottlenecks in circuits constructed by Conflux. Therefore, for our whole-network deployment experiments, we focus on browsing bridge users and bulk clients since those two classes of clients have more performance incentives to use Conflux.

Before we present our whole-network deployment results, we first compare the stock Tor download time measurements obtained from our testbed, when 370 web and 30 bulk clients are used, with the live Tor network measurements maintained by the Tor metrics portal [To12b]. Figure 3.11(a) shows that the stock Tor download time distribution, of 320 KiB files obtained using ExperimenTor, fits between the download time distributions of the 50 KiB and 1 MiB files

obtained from the live Tor network. Note that the Tor project only maintains the download times of 50 KiB, 1 MiB and 5 MiB file sizes over the live Tor network.

Second, Figure 3.11(b) demonstrates that our testbed's download time distribution of 5 MiB files closely approximates the respective distribution obtained from the live Tor network. In fact, the results look accurate for the fourth quartile of the download times, and for the first three quartiles, the testbed performance is only 15% slower than the live Tor network.

Although we believe that using 370 web and 30 bulk clients produces an accurate approximation, we also experiment with a lighter load of 390 web and 10 bulk clients, as there are continuous efforts to reduce the load in the network by throttling the bulk downloaders [JSH12,MWS11]. Furthermore, in each experiment, among the 390 to 370 web clients, we fix the number of bridge users to 50. We also run an additional five low-bandwidth routers that act as bridges.[11] Those bridges are neglected by non-bridge users.

The whole-network experiments indicate that Conflux offers significant improvement, particularly for slower circuits. Under a light load of 10 bulk clients, shown in Figure 3.12(a), Conflux significantly improves the performance for bulk downloads compared to stock Tor. For example, at the median, it takes approximately 85 seconds to finish downloading a 5 MiB file for the Tor clients, whereas with Conflux, it takes approximately 60 seconds.

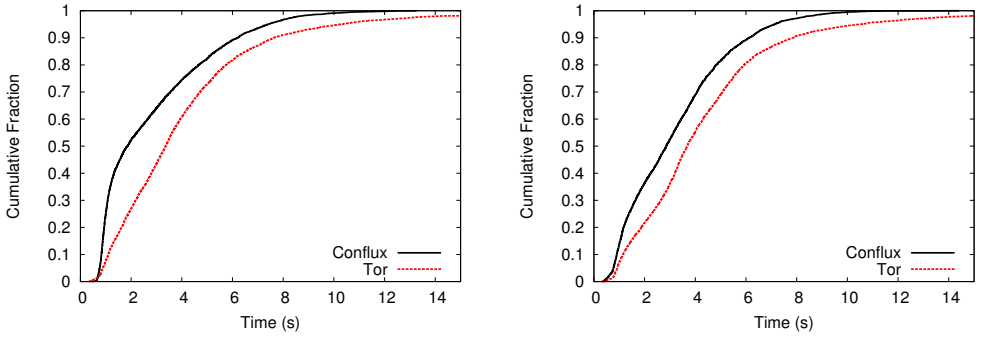
Under the regular load of 30 bulk clients,[12] as seen in Figure 3.12(b), Conflux and Tor offer a similar performance for 80% of the downloads as the amount of spare bandwidth in the network becomes less available. However, for 20% of the downloads, Conflux still outperforms Tor.

Next, we present the results for the web browsing bridge users. Figure 3.13 shows the significant performance gains in download times experienced by bridge users using Conflux when the network is lightly loaded with 10 bulk clients and 390 web browsers. At the median, it takes a Tor client 11 seconds to finish downloading the 320 KiB file, whereas with Conflux, it takes only 6 seconds, which is approximately a 45% improvement. When we increase the load to 30 bulk clients to match the performance of the current Tor network, we still observe significant download time improvements. For Tor clients, the download times are degraded by 4 seconds, as the download time reaches 15 seconds at the median, whereas with Conflux, the degradation is only 2 seconds, as downloads complete in 8 seconds. Therefore, even with congestion, Conflux maintains the performance advantage of roughly 46%.

Figure 3.14 tells us a similar story for the time-to-first-byte results. Under a light load of 10 bulk clients, Conflux needs only 1.8 seconds before the browser starts changing for the bridge

[11] Since little is known about the total number and behavior of bridge users, we make reasonable assumptions in designing these experiments.

[12] Note that it is usually assumed in the Tor literature that bulk downloaders constitute approximately 3% of all clients [MBG+08, MWS11].



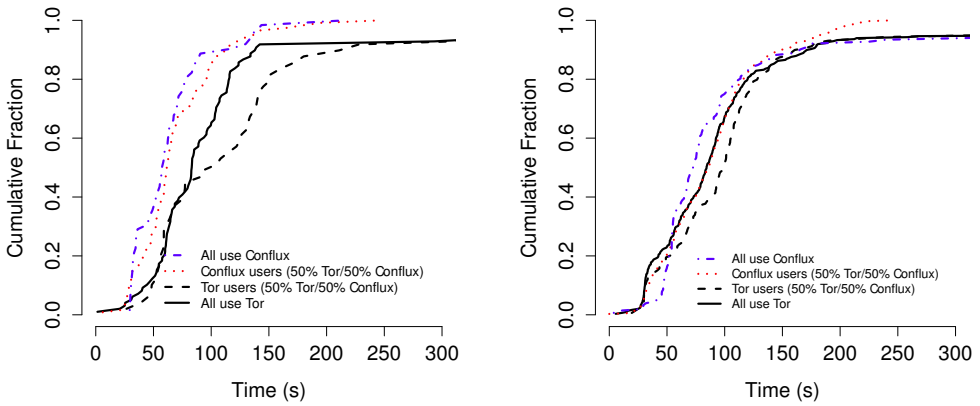
(a) 10 bulk and 390 web browsing clients (b) 30 bulk and 370 web browsing clients

Figure 3.14: Time-to-first-byte comparison between Tor and Conflux for web clients that download 320 KiB using bridges in whole-network deployment experiments

user at the median. The respective waiting time for the Tor client is 3.3 seconds, which is a 45% improvement in time-to-first-byte using Conflux. When we increase the load to 30 bulk clients, the median time-to-first-byte is still improved by roughly 23% when Conflux is used. Therefore, our large-scale experiments confirm our live experiments, in which we observed large performance benefits when Conflux is used for bridge users.

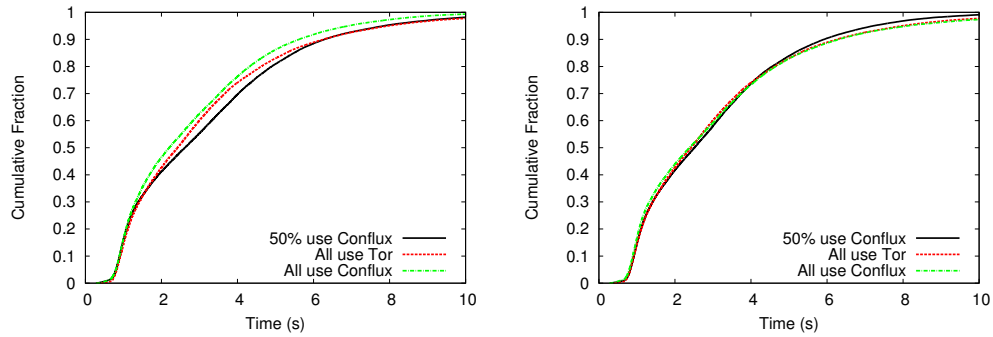
Partial deployment. We next present results for partial deployment experiments. It is important to understand how gradually upgrading the network with Conflux-enabled bulk clients might impact the performance of other clients. We modify our large-scale experiments so that half of our bulk clients run Tor while the other half run Conflux. The results are shown in Figure 3.15. As expected, the 50% Conflux-enabled clients outperform stock Tor users under different traffic loads. We also observe that web clients are unaffected by the partial Conflux adoption, across all traffic loads, as shown in Figures 3.16 and 3.17.

One interesting result that was revealed in this experiment is that when a fraction of the network uses Conflux, the performance of the rest of the bulk users in the network might get slightly worse compared to when no Conflux users are present in the network. The reason is that ORs will spend more time and bandwidth servicing the traffic of Conflux-enabled clients, which affects the performance of non-upgraded users. This is an incentive for clients to upgrade and obtain better performance. Tables 3.2 and 3.3 provide a concise summary of our results.



(a) 10 bulk and 390 web browsing clients (b) 30 bulk and 370 web browsing clients

Figure 3.15: Expected download times as different fractions of bulk downloaders adopt Conflux

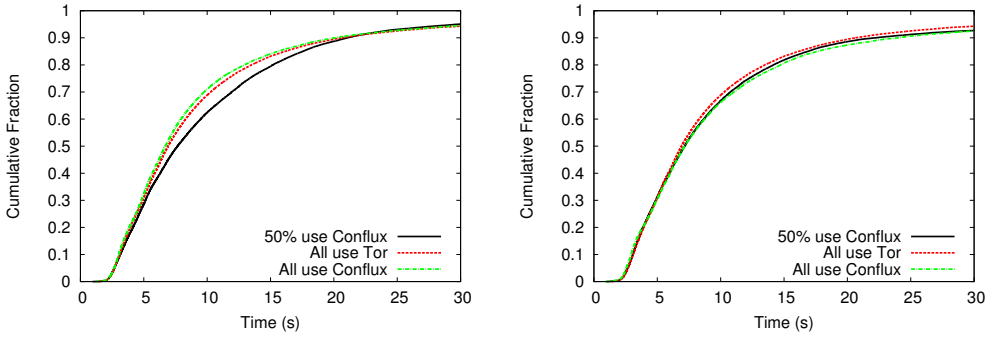


(a) 10 bulk and 390 web browsing clients (b) 30 bulk and 370 web browsing clients

Figure 3.16: Expected time-to-first-byte for web clients when bulk clients adopt Conflux

3.5 Security Analysis

We next investigate how the multipath routing scheme in Conflux affects the probability of the adversary linking together a circuit's source and destination. This linking, or *circuit compromise*, occurs when the adversary controls both endpoints of a given circuit [STRL00] and applies timing analysis [SW06, LRWW04] to reveal the communication patterns between the client and



(a) 10 bulk and 390 web browsing clients (b) 30 bulk and 370 web browsing clients

Figure 3.17: Expected download time for web clients when bulk clients adopt Conflux

Table 3.2: Relative download time comparison at the 80th percentile for Conflux and Tor under increasing traffic loads. The table summarizes Figures 3.12, 3.13 and 3.17.

User type	*Light load*	*Regular load*
Web browsing bridge users	Improved by 32%	Improved by 34%
Streaming users	Improved by 17%	Improved by 3%
Other clients	Not affected	Not affected

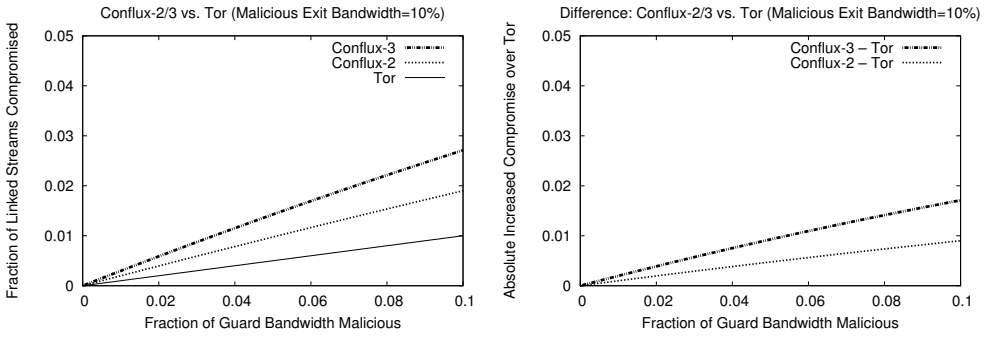
its corresponding destination. We first analyze the potential for path compromise due to passive attacks. Active attacks are considered in Section 3.5.3.

3.5.1 Identifying Bridge Users

Exit ORs can easily recognize which clients are using Conflux. Therefore, to prevent exit ORs from identifying bridge users, both bridge users and non-bridge users are encouraged to use Conflux. In fact, non-bridge browsing users can benefit from Conflux because browsing activity often involves streaming or downloading large files or images. In such situations, non-bridge users can observe between 3% to 17% performance improvements as we have seen in the results of our whole-network deployment experiments summarized in Table 3.2. This substantial improvement provides them with strong incentives to use Conflux and thereby aid in increasing the anonymity set of bridge users who adopt Conflux.

Table 3.3: Relative download time comparison at the 80th percentile for Conflux and Tor under increasing traffic loads for a partial deployment where 50% of clients adopt Conflux. The table summarizes Figures 3.15, and 3.17.

User type	*Light load*	*Regular load*
Upgraded streaming users	Improved by 32%	Improved by 5%
Non-upgraded streaming users	Degraded by 31%	Degraded by 7%
Other clients	Not affected	Not affected



(a) Path compromise for Conflux and Tor (b) Difference between Conflux and Tor

Figure 3.18: Security of Conflux with two and three circuits compared to stock Tor

3.5.2 Path Compromise

We next examine the effect Conflux has on client exposure and compare it to Tor. We look at both client compromise rates as well as individual circuit compromise rates to provide a more thorough discussion of the security implications of Conflux.

For client compromise, we adopt the model used by Elahi *et al.* [EBA+12], which provides a realistic and empirical approach to determining client exposure due to real-world network phenomena such as guard churn. In their model, clients are considered compromised whenever there is a malicious relay in their guard set. Note that if a client has a guard set containing malicious guards, then the number of circuits created before choosing a malicious guard would be sightly smaller for Conflux as compared to Tor. However, such a client would be deanonymized with either Tor or Conflux, and so from a *client compromise* point of view, we consider either situation to be equally bad. (We will consider *circuit compromise*, in which one cares about how *often* a client's circuits are compromised, below.) This captures the upper bound on the absolute

Table 3.4: Compromised circuit rates at different values of k (the number of entry guards used for a (multi)path) given m malicious relays in the user's guard set at 10% malicious guard bandwidth (for the computation of B_m) and 10% malicious exit bandwidth (for the computation of P_m)

		P_m(Compromised)		
Malicious Relays in Guard Set (m)	Probability of m (B_m)	$k=1$ (Tor)	$k=2$ (Conflux-2)	$k=3$ (Conflux-3)
0	72.9%	0%	0%	0%
1	24.3%	3.33%	6.67%	10%
2	2.7%	6.67%	10%	10%
3	0.1%	10%	10%	10%

compromise levels of the client population. The reason why only clients with all guards honest are assured safety is that if even one of the guards is malicious then the client will eventually pick a malicious guard at the same time as picking a malicious exit, and will be exposed.

This model depends only on the distribution of the guards amongst the clients' guard sets and is not affected by the multipath scheme Conflux utilizes. Conflux does not alter how guards are selected by clients into their guard sets nor how guards are selected for circuits. This makes the guards clients pick with Conflux identical to those picked with Tor, and hence both systems have identical exposure rates.

Under certain circumstances, individual circuit compromise rates can be just as important as client compromise rates. The following formula captures the probability of a compromised multipath in Conflux.

$$P(\text{Compromised}) \triangleq f_{xbw} \cdot (1 - (1 - f_{gbw})^k) \tag{3.1}$$

Here, k is the number of guard nodes[13] used in the multipath, while the adversary controls a proportion f_{xbw} of the network's exit bandwidth and a proportion f_{gbw} of the network's guard bandwidth. The equivalent value for Tor is $f_{xbw} \cdot f_{gbw}$, the probability of selecting both a malicious exit and guard. Note that, as expected, this is the same as the expression for Conflux when $k = 1$.

In order to understand the implications of this formula, we compare Conflux — denoted Conflux-2 where two guards are used on a multipath and Conflux-3 where three guards are used — with Tor and plot the results in Figure 3.18(a). It is clear that while Conflux-2/3 increases the chance of a compromised circuit it is very slight in absolute terms; at 10% compromised exit bandwidth it is 0.9% in Conflux-2 and 1.72% in Conflux-3, as shown in Figure 3.18(b).

[13]This analysis is also applicable to bridge relays since Conflux utilizes them in the same manner as guard relays.

The tradeoff between this slight increase and that of increased performance can be evaluated depending on the needs of the client.

We further analyze the probability distribution (B_m) of a client having m malicious guards in its guard set as well as the probability (P_m) of constructing compromised circuits given m compromised guards. Assuming 10% malicious exit bandwidth, Table 3.4 provides the probabilities of picking 0 or more malicious guard relays as well as the probabilities of constructing malicious circuits under those conditions. *Note that because malicious guards are chosen infrequently, Conflux often adds no additional vulnerability to path compromise over and above Tor.*[14]

The same analysis follows for AS-level adversaries that can observe the connections between clients and their guards/bridges, and between exit relays and the destinations. Substituting the fraction of circuits observed by the adversary between the two end points, i.e. $f_{client-guard}$ and $f_{exit-destination}$, for f_{gbw} and f_{xbw} respectively in Equation 3.1, we can calculate compromise rates in similar fashion to those in Table 3.4. While it is difficult to estimate realistic adversarial AS coverage, Edman and Syverson [ES09] and Wacek *et al.* [WTBS13] provide values of between 18–25% for the parameters above.

3.5.3 Selective Denial of Service

We now consider active attacks, in the form of selective denial of service (SDoS) [BDMT07, BMG+07, DB13, THK09]. Under this attack the adversary actively breaks circuits he is part of if he finds that either end point is not controlled by him. This causes the client to create compromised circuits at a higher rate. The adversary can choose to either SDoS circuits immediately or be strategically patient.

Recall that Conflux establishes *primary circuits* first to which further (secondary) circuits are linked to form multipaths. Hence, this mechanism provides two avenues for the SDoS attack; first at the primary circuit building stage and second at the circuit linking stage. We analyze each in turn.

We model the likelihood of primary circuit compromise with SDoS by inputing Formula 3.1 from Section 3.5.2 in the following formula, proposed by Das and Borisov [DB13], where f is the fraction of malicious relay bandwidth:

$$P(SDoS) = \frac{P(\text{Compromised})}{P(\text{Compromised}) + (1-f)^3}$$

[14] As of July 2012, the Tor network had roughly 1,200 MiB/s aggregate guard bandwidth and roughly 600 MiB/s aggregate exit bandwidth [To12a]. We believe that an adversary in possession of 10% of each of these bandwidths is a powerful, high-resource attacker. Thus, this should be considered a worst-case security analysis.

With SDoS, constructed primary circuits will either be compromised ($P(\text{Compromised})$) or entirely honest ($(1-f)^3$). At 10% malicious bandwidth this attack increases the likelihood to 2.54% from 1.9% for Conflux-2 and to 3.57% from 2.7% for Conflux-3. Compare this to Tor where the same attack increases the likelihood to 1.35% from 1%. The increases here are due to the primary circuit creation phase and not due to Conflux's multipath linking scheme.

The strategic adversary servicing a primary circuit with a compromised exit relay may gain an added advantage by performing SDoS exclusively on the secondary circuits being linked to the primary. At 10% malicious bandwidth the probability of this scenario is 9%. Under Conflux-2, employing SDoS on such secondary circuits exclusively gains the adversary 0.23% whereas Conflux-3 gains the adversary 0.41%. The adversary needs to balance the cost of this strategy in terms of bandwidth used to service the uncompromised primary and the added advantage it provides.

A possible countermeasure is to ensure that the OP will retry the linked circuit using the same guard and will not switch to another (potentially malicious) guard. After a few retries, if it is unable to build a *secondary circuit* for the linked stream, the OP will give up but without impacting the client whose traffic still flows over the uncompromised primary circuit. Note that Tor is currently working on preventing the SDoS attacks by allowing clients to maintain several variables for each of its entry guards in order to detect the SDoSing guards [DM13]. If the Tor client is able to identify the SDoSing guards, then it can avoid them. If the client only uses honest guards, then it is safe from SDoS for both Tor and Conflux.

3.6 Discussion

We next discuss a variety of open issues with our design and describe our future work.

3.6.1 Congestion

Conflux does not introduce any additional traffic to the network as our multipaths are bounded by the window of the primary circuit and by the bandwidth of the exit routers. Indeed, in our large-scale experiments, we found that the windows are not exhausted when Conflux is used. As indicated by our results, congestion is reduced because Conflux dynamically senses latency on each circuit as a congestion indicator, which allows our traffic splitting approach to perform smart load balancing so that congestion can be avoided. Moreover, Conflux is complementary to incentive schemes that seek to increase the number of exit routers and bridges.

3.6.2 Computational Cost

One might wonder if Conflux adds more complexity to the operation or routers since it requires building more circuits resulting in more public key (PK) operations. Since Tor clients build spare idle circuits by default, Conflux can use those circuits as secondary circuits in order not to add more PK operations in building multipaths. For circuits that must be built on demand, Conflux doubles the cost of building circuits (assuming only two paths are used). However, those costs can be significantly reduced if Conflux uses Tor 0.2.4's ntor handshake [DM13], which eliminates PK decryptions at routers during circuit construction.

3.6.3 Experimental Limitations

The results we obtained show an improvement in a testbed environment. In our experiments, we emulate the network behaviour and traffic load using ExperimenTor, since it allows us to set up different network topologies with different bandwidth and link settings. For practical purposes, we scaled the network down to 20 routers, which is not reflective of the real Tor network. However, we strived in our experiments to perform and obtain realistic performance measurements under different traffic loads. Performing large-scale experiments using different network models such as the ones recently proposed by Jansen *et al.* [JBHD12] and Wacek *et al.* [WTBS13] and performing a large-scale performance evaluation on the live Tor network are areas for future work.

3.6.4 Future Work

Recall that our dynamic traffic splitting technique is based only on two circuits. One area we want to explore is using Conflux with more than two circuits. Since we found the performance improvements to be significant with two circuits, we suspect that we may find even more performance benefits with more than two circuits. We leave exploring the performance benefits of splitting traffic over more than two circuits for future investigation.

Another avenue for future work is to evaluate alternative methods for measuring circuit latency. Recall that we opportunistically measure the latency of a circuit in order to access its performance using Tor's existing circuit-level `SENDME` cells that a client sends to the exit router after receipt of every 100^{th} data cell. This allows us to measure circuit latency/congestion for free (e.g., with no additional overhead to the network). An alternative technique is to introduce a new cell type that clients send to exits and perform more frequent measurements. A potential advantage of this approach is that it might detect sudden changes in latency and thereby react

to congestion faster. However, a downside is that this approach is more hostile to the network, because it places more probe traffic onto the network.

3.7 Related Work

Load balancing and multipath for IP and peer-to-peer networks. Multipath routing is a well-studied problem in the networking literature. Common core network routing protocols, such as Open Shortest Path First (OSPF) [Moy98] and Interior Gateway Routing Protocol (IGRP) [Hed91], implement load balancing, where routers distribute traffic among several possibly disjoint paths that have equal (or unequal) costs. These routing protocols measure the cost of each path and distribute traffic accordingly. The purpose is to relieve congestion and cope with failures. Another example is BitTorrent [Coh11], which is a peer-to-peer file-sharing protocol that utilizes multipath routing to quickly and efficiently distribute files. BitTorrent peers maintain many simultaneously active connections with other peers, using an rarest first data request strategy with an "optimistic unchoking" algorithm to mitigate selfishness. Finally, MPTCP [HSH+06] is a transport layer protocol that allows applications to send data over several interfaces/addresses. MPTCP is more useful if divergent paths are available (multi-homing).

In the above mulitpath examples from IP networks, routing decisions are done by IP routers, and a TCP source cannot choose the Internet path. Conflux, on the other hand, is implemented at the overlay layer. Because data is source-routed in Tor, Conflux chooses the multi-paths and ensures they are only joined at the exit.

Multipath for anonymity networks. In the context of high-latency Chaumian mix networks, Serjantov and Murdoch [SM05] show that sending packets over multiple disjoint routes through the mix network may increase anonymity against a partial passive adversary.

In the context of onion routing networks, MORE [LPW+07] routes every packet through a different path of onion routers, half of which are chosen by the sender and half chosen by the receiver. While this approach offers the ability to create highly dynamic paths—which can have desirable performance, load balancing, and anonymity properties—the requirement that communicating parties participate in the anonymity network reduces MORE's practicality.

To increase throughput in Tor, Snader [Sna10] presents preliminary experiments in which clients receive n different streams over n disjoint circuits. In particular, multiple circuits are shown to reduce the time to download 1 MB files. However, it is unclear whether performance is improved for smaller web-like streams; furthermore, since this scheme uses more entry and exit points into the Tor network, it may increase the threat of end-to-end traffic confirmation attacks (as in Bauer *et al.* [BMG+07]).

Although Snader's experiments have shown promising initial results for multipath routing, his study is different from ours in its goals, design and experimental approach. Our goal is to improve load balancing, and reduce download times and congestion, whereas Snader focuses mainly on throughput. Also, our technique encourages using resource-constrained routers, whereas Snader's experiments reveal a degradation in throughput when low-bandwidth circuits are used. Furthermore, Snader's design uses multiple disjoint circuits to download different parts of the same file to improve throughput. In contrast, our multipaths intersect at the exit node, which gives us two key advantages. First, it allows us to implement a dynamic traffic splitting algorithm that assigns different loads to circuits according to their throughput. Second, unlike Snader's approach, our design is application-agnostic. We do not require external servers to be capable of serving different pieces of files to different TCP connections. In addition to web browsing, our design supports any TCP-based application, such as `scp` and file transfers. We also conduct whole-network and live experiments to show the performance benefits of our approach.

Multipath routing has also been proposed to mitigate timing attacks [SW06, SS03] in low-latency anonymity networks. Feigenbaum *et al.* [FJS10] introduce a new anonymous communication structure, called a layered mesh topology, that defends against such attacks without delaying the user's traffic (as in mix networks). In the layered mesh, circuits carrying data from the client to a destination server are composed of w paths, where every path consists of ℓ relays, and all paths intersect at a common final (exit) router. The key difference between this work and Conflux is that Conflux aims to use multiple circuits to increase performance; in contrast, Feigenbaum *et al.* focus only on defeating timing attacks.

Combination with other Tor performance work. As discussed in Chapter 2, there are several proposals that aim to improve the performance of Tor in different ways such as congestion control and avoidance [TG10, RG09, WBFG12], improved router selection [SB08, SBL09], scalability [MTHK09, MOT+11], and applying incentive schemes to increase the number and bandwidth of Tor relays [JHK10, NDW10, MWS11]. Conflux is complementary to these previous approaches and we expect that even greater performance benefits are possible by combining these proposals together. We leave this for future investigation.

3.8 Conclusion

This work is motivated by the need to improve performance for Tor clients who utilize low-bandwidth bridge nodes to access the Tor network from locations that block access to Tor. We presented the design, implementation, and analysis of Conflux, a dynamic multipath traffic splitting scheme that offers higher throughput service even when clients utilize low-bandwidth nodes.

Conflux is easy to deploy on the current Tor network, as it only requires upgrades for Tor clients and exit routers.

Our whole-network performance evaluation demonstrates a potential for decreased latency and increased throughput with our scheme, particularly for low-bandwidth bridge users. Interestingly, we also observed that users of emerging web applications such as streaming video also benefit from Conflux. Furthermore, experiments conducted on the live Tor network show that real Tor users today could experience better performance with Conflux. Our hope is that this work improves Tor's performance and usability, leading to a larger user base and greater anonymity properties.

Chapter 4

Enhancing Tor's Performance using Real-time Traffic Classification

An earlier version of this chapter appeared in the 2012 ACM Conference on Computer and Communications Security [ABG12].

In this chapter, we seek to improve the performance of Tor by defining different classes of service for its traffic. We recognize that although the majority of Tor traffic is *interactive web browsing*, a relatively small amount of *bulk downloading* consumes an unfair amount of Tor's scarce bandwidth. Furthermore, these traffic classes have different time and bandwidth constraints; therefore, they should not be given the same Quality of Service (QoS), which Tor offers them today.

We propose and evaluate DiffTor, a machine-learning-based approach that classifies Tor's encrypted circuits by application *in real time* and subsequently assigns distinct classes of service to each application. Our experiments confirm that we are able to classify circuits we generated on the live Tor network with an extremely high accuracy that exceeds 95%.

We show that our real-time classification in combination with QoS can considerably improve the experience of Tor clients, as our simple techniques result in a 75% improvement in responsiveness and an 86% reduction in download times at the median for interactive users.

4.1 Introduction

Recall that one of the main causes of performance degradation in Tor is the existence of greedy applications, which can consume a high percentage of the network bandwidth. To reduce the congestion caused by greedy applications, global or static throttling techniques have been introduced [MWS11,JSH12] that limit the rate at which these applications can transfer data. There are a number of problems with global network throttling techniques, however. First, they are either based on fixed throttling rates that can also affect interactive or real-time application users, or their ability to detect greedy applications is based on simple metrics that are easy to game. Also, we believe that global network throttling prevents the network from optimizing its resources effectively.

In this chapter, we propose defining classes of service for Tor's traffic. Our solution is based on the key observation that different applications that use Tor have inherently different time and throughput requirements, and therefore should not need or get equivalent service, which Tor offers them today. By defining different classes of service, we can map each application class to its appropriate QoS requirement. This solution is significantly more flexible than previous approaches and it provides relays with more dynamic control over their limited resources. For example, it allows relays to prioritize or throttle one application class over another. When the prioritized class ceases communication, the throttled class can utilize the available bandwidth. Furthermore, even though the network has limited capacity, this approach allows for finer-granularity management of the existing network resources. Such management is needed to provide good performance to important applications.

Classification for Tor. Motivated by the need to improve the performance of the Tor network, we introduce DiffTor, a framework for classifying encrypted Tor circuits based on the applications they serve. The key novelty in our work is that we enable differentiated QoS using network traffic classification in anonymous communication systems. We explore two types of traffic classification: online and offline classification. Below we highlight the differences between the two classification types:

- **Online:** The main goal of online classification is for a router to classify circuits on the fly in order to provide differentiated services. This means that classification must take place while the circuit is alive, and before too much traffic has been forwarded along the circuit. Online classification is more challenging since the classifier has access to less information about the circuit. The classifier performs classification at the cell level and is mainly concerned with cell-level attributes and some limited circuit-level attributes. Also, since the classifier runs in real time, it must also be efficient.

- **Offline:** On the other hand, classification can be done offline by relay operators using circuit logs (assuming that logging is performed in a privacy-preserving manner). Because there are no time constraints in offline classification, we can use more global information about circuits as classification attributes.

The importance of offline classification is twofold. First, it is important to understand how much information an entry guard—the first router in a Tor circuit—can infer just from monitoring the traffic of its clients' circuits. This is especially important to investigate about entry guards since they receive direct connections from clients and continue to be used by the same clients for weeks. Indeed, we found that entry guards can identify the class of application of a circuit with an accuracy that exceeds 90%. Second, The Tor Project is researching techniques to understand more information about its users without compromising their privacy [Loe10]. Classifying circuits offline is a novel approach to provide more insight into how the network is used today from the view of entry nodes unable to see plaintext traffic.[1]

Next, we consider online classification. Classifying Tor circuits on the fly allows us to define Quality of Service (QoS) requirements for different types of traffic. Defining QoS parameters can be done either locally by the entry guard operator or globally by the directory authorities depending on the state of the network. For example, when the network is congested, entry guards can change their QoS parameters so that interactive real-time applications such as web browsing are prioritized, while bandwidth-consuming applications such as non-time-sensitive file sharing applications are throttled. This can improve the performance of the network and the experience of users, since different applications would get different QoS parameters based on their requirements and on the congestion state of the network.

In this chapter, we carry out the following approach. We first start by exploring how different applications use the Tor network. In particular, we seek to understand how the traffic of some popular applications looks and behaves when travelling through Tor circuits. The importance of this step is that it helps us derive useful attributes that we can use for our online and offline classification process.

Based on studying different applications, we first start by developing a simple threshold classifier that we can use to classify connections between clients and entry guards to two applications, BitTorrent and browsing. We recognize that this classifier cannot be used on middle and exit relays, and would not be useful if clients disable using entry guards. For these reasons, we explore different machine-learning classifiers and identify which algorithms are suitable for our offline and online classification, which unlike the threshold classifier, can also work at middle and exit

[1] Previous work used Deep Packet Inspection (DPI) to study the usage of the Tor network from exit routers [CMK10, MBG+08]. It is important to avoid inspecting the clear traffic for legal and ethical reasons.

routers in addition to entry guards. Finally, we implement an online classifier and we define simple QoS rules to show the performance benefits made possible using our online classifier.

Contributions. This work contributes the following:

- We provide an important insight into how some popular applications behave and appear within Tor circuits.
- We propose a machine-learning approach to encrypted traffic classification for Tor. Our evaluation on the live Tor network indicates that our classifiers achieve over 95% accuracy. When combined with prioritization for interactive web-browsing circuits, we see an improvement in responsiveness of 75% and an improvement in download time of 86%.
- Our approach to prioritization is incrementally deployable, as our changes are local to any router and do not affect its operation with other routers.
- We discuss a variety of open issues related to the classification of encrypted Tor traffic.

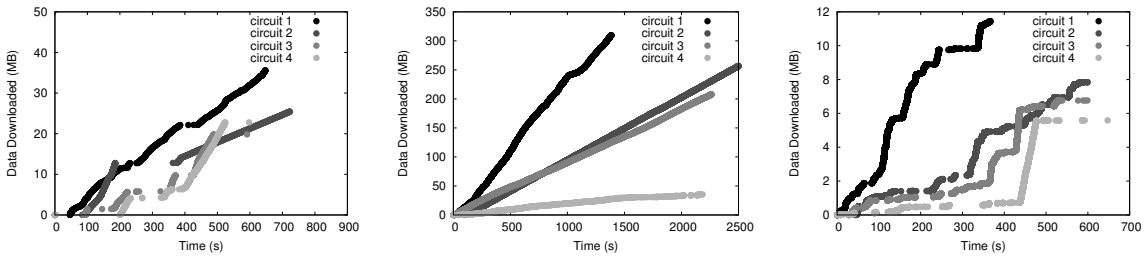
Roadmap. The remainder of this chapter is organized as follows: Section 4.2 illustrates how different applications use Tor. We present our threshold classifier and motivate for the need for machine-learning-based classifiers in Section 4.3. Section 4.4 presents our three broad classes, our candidate features and our machine learning classification methods. We evaluate our proposal in Section 4.5, and we discuss a variety of open issues related to the classification of encrypted Tor traffic in Section 4.6. Finally, we compare our contributions with related work in Section 4.7 and conclude in Section 4.8.

4.2 How Different Applications use Tor

We first seek to explore how different applications affect how the traffic looks on circuits. This is important for us to understand in order to be able to nominate some useful attributes for our classification process.

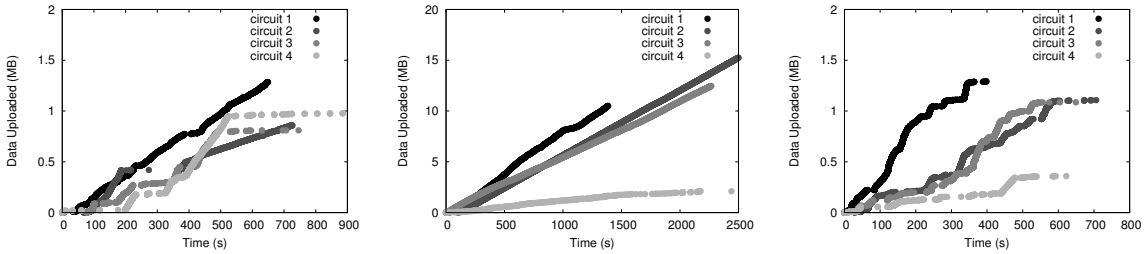
Figures 4.1 and 4.2 show data transferred upstream and downstream on some sample active Tor circuits that we logged on the live Tor network. The traffic was generated by our own browsing, streaming, and BitTorrent clients; for privacy reasons, we did not monitor other users' Tor traffic. We provide a detailed description on how we generate and log traffic on the live network in Section 4.5.1. The following differences can be observed from the figures.

Applications produce different circuit lifetimes. The circuit lifetime is different for different applications. Web browsing circuits almost never exceed 600 seconds, which is the default circuit



(a) Streaming (b) BitTorrent (c) Browsing

Figure 4.1: Downstream circuits



(a) Streaming (b) BitTorrent (c) Browsing

Figure 4.2: Upstream circuits

lifetime, while actively downloading BitTorrent circuits can stay alive for as long as the client is actively downloading on the circuit. In fact, it can be seen in Figure 4.1(b) that BitTorrent downloads can cause the circuit to stay alive for more than 2500 seconds. Streaming circuits can exceed the 600-second mark, but once the streaming session is over, the circuit will be torn down.

BitTorrent traffic is bidirectional. In active BitTorrent circuits, data is continuously travelling upstream and downstream. Web browsing and streaming circuits on the other hand appear to have bursts of data that are followed by gaps of inactivity. Those idle periods may be a result of user think times. However, some web browsing circuits also seem to be continuously active due to the dynamic content of web pages.

Interactive circuits transfer small amounts of data. More than 95% of circuits in each class download less than 14 MB, as shown in Figure 4.3. However, while the maximum observed download for any streaming circuit was less than 35 MB and for any browsing circuit was less than 23 MB, some BitTorrent circuits downloaded as much as 935 MB.

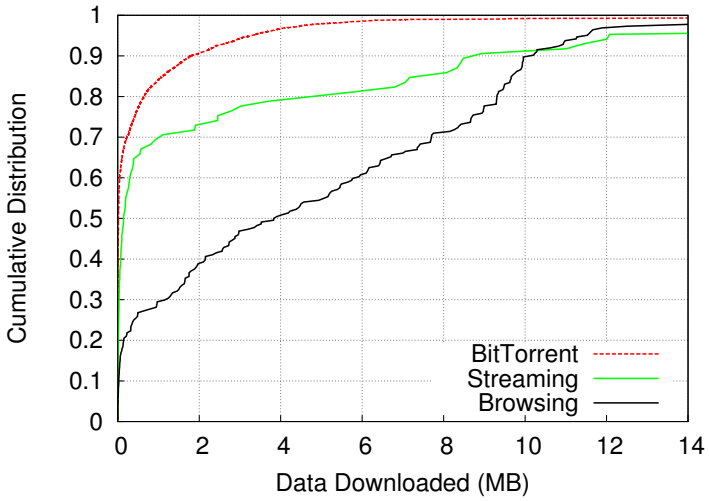


Figure 4.3: Comparison of the amounts of data downloaded by circuits of different applications

BitTorrent causes many idle circuits to be built. Finally, as shown in Figure 4.3, most circuits created for web browsing are indeed utilized, while only 30% of the BitTorrent circuits are actually used to download any significant amount of data. We observed that the BitTorrent client creates many more circuits than a browsing client. The reason is that the BitTorrent client attempts to contact several peers at the same time using different ports. Those circuits attempt to connect to peers that either are no longer participating in the file transfer, or do not respond to connection requests due to the presence of a firewall or network address translation (NAT). Interestingly, when a remote host fails to respond to a connection request (and does not respond with "connection refused"), Tor's default behaviour is to retry the same remote host again on a different circuit. This behaviour is rarely triggered by the browsing client since it usually connects to a few publicly accessible servers.

Creating several idle circuits has performance and anonymity implications. Clearly, circuit creation consumes OR memory and CPU cycles. As for anonymity, creating several circuits simultaneously can degrade the anonymity of the user as it can leak some information to the entry guard about the user. For example, we observed that in a 10-hour experiment, our web browser created around 100 circuits, while the BitTorrent client created more than 1000 circuits. If an entry guard observes that the user is using several simultaneous circuits, then very likely, the client is using BitTorrent over Tor.

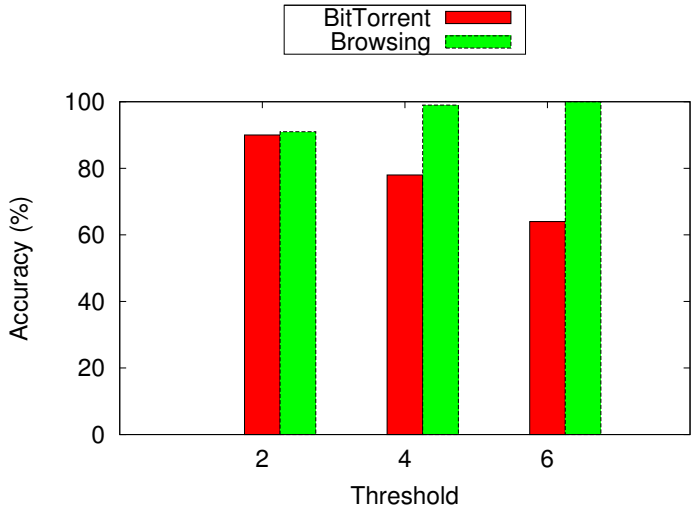


Figure 4.4: Per-class accuracy of the threshold classifier when $t = 100$

4.3 Straw Man: Circuit Threshold Classifier

Recall that BitTorrent causes the Tor client to build significantly more circuits compared to other interactive applications, which usually do not require more than one circuit at a time. In this section, we show how an entry guard can use a simple threshold classifier to perform a connection-level classification on its clients' connections to two broad classes, BitTorrent and web browsing. The classifier's decisions are based only on the number of circuits created over a connection.

Our simple classifier monitors the clients' connections, and performs a classification periodically every t seconds based on the number of circuits T seen in the connection within the last t seconds. Figure 4.4 shows the per-class accuracy as a function of the threshold value when $t = 100$. If the threshold is $T > 2$, meaning that if we observed more than two circuits in the connection within the 100-second interval, then the probability that the connection serves BitTorrent is 90%. Also, with 91% certainty, if the connection carries two circuits or fewer, then the connection is serving web browsing. As we increase the threshold value to $T > 6$, we get perfect 100% classification accuracy for web browsing connections at the expense of misclassifying 36% of BitTorrent connections as browsing connections.

Although this classifier is very simple and requires no prior collection of any training data, it has several limitations. First, this classifier works at the connection level, and therefore, we cannot use it at intermediate ORs in a circuit since the OR-to-OR connections multiplex circuits from several users. Second, a client can easily game the classifier by not using entry guards. That

will significantly reduce the number of circuits observed by one guard for a specific user. In the next section, we show how we avoid these problems by using machine learning approaches to classify Tor traffic at the circuit and cell levels.

4.4 Classification for Tor

In this section, we start by defining the three traffic classes we are interested in. Then, we describe the attributes and the machine learning algorithms that we use for the online and offline classification.

4.4.1 Class Definition

A previous Tor traffic study [MBG+08] reported that web browsing traffic results in over 92% of the TCP connections in the Tor network, yet it accounts for only 60% of the aggregate traffic volume. The other 40% of the aggregate traffic volume is mainly accounted for by BitTorrent. Based on these findings, we define three broad classes of traffic:

Bulk transfer. This class contains applications that upload or download large volumes of data over the network. Such downloads have no time constraints, but are greedy and will try to utilize as much available bandwidth as possible. BitTorrent is the main application in this class[2] since it accounts for a great portion of the traffic in Tor.

Interactive. Interactive applications are real-time applications that require interaction between a client and a server (or another client in the case of instant messaging, for example). Those applications are time sensitive and require improved responsiveness. We focus on web browsing as the main application of this class.

Streaming. Streaming applications are non-interactive bulk transfers that require time and quality constraints. An example for streaming applications is watching streaming videos online.The reason we consider streaming applications is that recent Internet measurement studies reported a significant increase in the amount of streaming traffic online [MFPA09, Sa11].

[2] Although BitTorrent is switching to UDP, recent studies in Tor suggest that there is still a significant amount of BitTorrent traffic served by Tor [BMC+11].

4.4.2 Candidate Attributes

Attribute selection is a non-trivial problem in the area of network traffic classification. It gets particularly more difficult for an anonymity network like Tor where we do not have access to traditional useful attributes such as the connection ports and the IP packet sizes. We also take the challenge further as we try to realize real-time classification. In selecting candidate classification attributes for classifying Tor's traffic, we strive to identify high-level protocol features that are not influenced by human interaction variability. Below we present our attributes.

Circuit lifetime. By default, if a circuit has been used for 10 minutes and is currently idle, the Tor client tears down the circuit and starts attaching new streams to different circuits. Recall that we found in Section 4.2 that bulk downloading applications such as BitTorrent cause circuits to remain alive well beyond the 10 minute mark.

Data transferred. The amount of data that a circuit transfers upstream and downstream can be very indicative of the type of application that is being used by a circuit. For example, circuits that serve bulk downloads are expected to transfer significantly more data than circuits that serve web browsing sessions. Moreover, the amount of data uploaded to destinations outside the network can indicate that the circuit is serving a bulk transfer rather than an interactive application.

Cell inter-arrival times. The time between two consecutive cells in one circuit (and other related features such as the mean and variance of cell inter-arrival times (CIT)) can also indicate the type of application that uses the circuit. For example, bulk downloading applications such as BitTorrent almost never pause throughout their downloads. Interactive applications, on the other hand, consist of smaller bursts of data which are separated by gaps on inactivity. CIT statistics are useful in quantifying the *burstiness* of the traffic. Burstiness is a key attribute in separating different classes of traffic because it is a high-level attribute that is not influenced by the behaviour of individual users, but rather by how different application protocols work. Measuring the cell inter-arrival times can be done by subtracting the arrival times of two consecutive cells. To calculate some statistics of the inter-arrival times efficiently, we use the following recursive estimators [Kar06]:

$$\mu_t = \frac{t-1}{t}\mu_{t-1} + \frac{1}{t}x_t \tag{4.1}$$

for the average, where t starts at 1, x_t is the current measurement average, and μ_{t-1} is the previous average. Likewise, the variance can be computed as follows:

$$\sigma_t^2 = \frac{t-1}{t}\sigma_{t-1}^2 + \frac{1}{t-1}(x_t - \mu_t)^2 \tag{4.2}$$

where t starts at 2, x_t is the current measurement data, μ_t is the current average, and σ_{t-1}^2 is the previous variance. In addition to μ_t and σ_t^2, we also use the *coefficient of variation* as an attribute, which is given by $\frac{\sigma_t}{\mu_t}$.

The number of cells sent recently. Normally, it is expected with bulk downloads that the average number of cells seen recently would be consistently large, while interactive applications usually have smaller bursts of data followed by a period of inactivity. To quantify this feature, we use the Exponential Weighted Moving Average (EWMA) of the cells sent on a circuit in an algorithm that was proposed by Tang and Goldberg [TG10].

4.4.3 Classification Algorithms

Machine learning techniques are categorized as *supervised* or *unsupervised*. Supervised machine learning techniques are based on two stages. The first stage is known as the *training* stage where a classification model is built using a classification algorithm and a training dataset that contains labelled instances. The second stage is known as the *testing* stage in which the classifier model obtained from training is used to classify unlabelled instances.

Unsupervised techniques, on the other hand, seek to group or cluster data together based on some similarities, or common characteristics, without knowing the classes beforehand.

In this section, we provide a brief description of the four supervised classification algorithms we use in this work; we focus on the Naïve Bayes classifier since we implemented it for our live experiments:

Naïve Bayes. Naïve Bayes is a probabilistic classification algorithm that is based on Bayes' theorem.

$$p(C|A_1, A_2, .., A_n) = \frac{p(C)p(A_1, A_2, .., A_n|C)}{p(A_1, A_2, .., A_n)} \tag{4.3}$$

Assume that a circuit X has attributes $A_1, A_2, .., A_n$. For example, A_i might denote the circuit lifetime, or the number of cells sent. We would like to find the probability that X belongs to a class C, which is one of a finite set of classes. In our context, those classes are bulk, interactive and streaming. If the attributes are dependent, then computing the above conditional probability would be difficult. However, if we make the "naïve" assumption that the attributes are independent, then, our conditional probability can be expressed as follows:

$$p(C|A_1, A_2, .., A_n) = \frac{p(C)\prod p(A_i|C)}{p(A_1, A_2, .., A_n)} \tag{4.4}$$

Note that $p(A_1, A_2, .., A_n)$ can be ignored as it is constant for all $p(C_i|A_1, A_2, .., A_n)$. The rest of the right-hand side of the above expression is determined from the training data. Since

attributes in our case are numerical, a preprocessing step of discretization is needed to be able to compute $p(A_i|C)$. We have made the simplifying assumption that our data follows a normal distribution, so $p(A_i|C)$ can be computed using the normal density function. The last remaining step for the classifier is to make a decision rule based on the $p(C_i|A_1, A_2, .., A_n)$ values it computes for all classes. One common decision rule is to choose the most probable class.

Although the Naïve Bayes classifier is based on strong assumptions of feature independence, it is known to work well in practice, even if this assumption is violated. Furthermore, we chose to evaluate Naïve Bayes due to its strong performance in several related encrypted traffic classification tasks [LL06, HWF09].

Bayesian Networks. Bayesian Networks (or Bayes Nets) have been found to be very successful at classifying Internet traffic by application [AMG07]. Bayes Nets are graphical representations that are used to model the dependency relationships between attributes and classes; therefore, unlike Naïve Bayes, they do not make the strong assumption that attributes are independent. They consist of a a directed acyclic graph (DAG) where the vertices represent attributes and classes. The edges between vertices represent the dependencies between the attributes and classes. Also, the Bayes Net contains conditional probability tables that allow for probabilistic inference. Friedman *et al.* present Bayes Nets in more detail [FGG97].

Decision Trees. We also use two types of decision tree classifiers. We apply functional tree (FT) classification [Gam01], which allows nodes in decision trees to represent multiple features. We also apply logistic model tree (LMT) classification [LHF05]. We choose to evaluate a representative set of decision tree classifiers due to the success of similar classifiers in classifying multiple applications within IP packets traces [EBR03].

4.5 Experiments and Results

Our experiments were carried out as follows. First, from the live Tor network, we collected traffic logs, from which we created datasets that contained instances of the attributes we presented in Section 4.4.2, along with their respective classes. Then, we evaluated our online and offline classification algorithms on the datasets using the Waikato Environment for Knowledge Analysis (WEKA) software suite [HFH+09]. The WEKA software[3] supports a collection of machine learning algorithms, together with preprocessing functionalities that facilitate preparing our datasets for training and testing. Next, we fed the training model we obtained from WEKA into the simple Naïve Bayes classifier that we implemented in Tor in order to perform classification on the fly.

[3] We used version 3.6.6 for all our experiments.

Experimental setup. To collect training data, we configured three client types: a BitTorrent client, a web browsing client and a streaming client. All our clients run from a single machine and are forced to choose an OR under our control as an entry node for all their circuits, as we performed our measurements at our entry node.[4] Next, we describe how we set up our clients and we explain how we carry out our measurements safely without endangering the privacy of other users.

Automating the web browsing client. In order to generate realistic browsing traffic, we use the iMacros Firefox plugin [iMa12] to automate the process of web browsing. The automated browser works as follows: First, it picks a random URL from the list of the top 100 URLs reported by Alexa [Ale12] and starts downloading the web page. Then, after the page loads, the browser waits for a random amount of time that is selected from the distribution of user think times [HCJS03]. If the URL chosen is a search engine, then the browser types a keyword, which is also randomly selected from the top 100 search terms reported by Alexa, in the search box. Finally, after the results are loaded, the browser follows a random link from the top 5 search link results. This process is repeated in a loop of several iterations.

BitTorrent client. In order to generate P2P file sharing traffic, we used the Vuze BitTorrent client [Vuz12], which allows us to configure the proxy port on which our Tor client is listening. Note that Vuze contains some explicit options to use the Tor network for tracker and file transfer downloads and uploads. We used these options to send all our Vuze client traffic through Tor. We capped the maximum upload and download rates to 2000 KB/s. During our experiments, we noticed that our BitTorrent client easily achieves around 1000 KB/s. We sampled our torrent files from popular legal torrent websites. Our downloads included music, movies, and some Linux distribution files.

Streaming client. Again, we used the iMacros Firefox plugin in order to automate a streaming client. The client searches for videos using a random keyword selected from the top 100 search keywords reported by Alexa. Then the client selects a random video link from the returned results and watches for a random time between 1 to 5 minutes.[5] This wait time captures scenarios where the client watches part of the video and then either browses away to another page, or chooses to finish watching the whole video before searching for another keyword. Note that the streaming activity includes some browsing activity as well.

MeasureMe cell. Because we collect data and traffic statistics on the live Tor network using our entry OR, it is important to ensure that we do not log statistics of other users' traffic as this might reveal their private information. To achieve this goal, we implemented a new relay command cell type, which we call the "MeasureMe" cell. This cell is sent from the client to any OR on the

[4] During the process of data collection, we capped our node's bandwidth to 500 KB/s.

[5] User think times are known to be larger for YouTube sessions than for traditional web workloads [GALM08].

circuit to instruct it to start logging measurements on that particular circuit. To help distinguish which application uses which circuit, we configured each client with a different MeasureMe ID, which is sent to the measuring OR in the MeasureMe cell.

In our experiments, when each client starts building circuits, it will also send a MeasureMe cell to our entry node after a circuit is fully constructed. On receiving the MeasureMe cell, our node starts logging the classification attributes we described in Section 4.4.2 for the respective circuit.

4.5.1 Data Collection

Over a period of 6 weeks, from early March to mid-April 2012, our BitTorrent, automated browsing and streaming clients downloaded approximately 24 hours' worth of traffic. Because we logged the MeasureMe ID along with circuit id and its attributes, we were able to divide our traffic logs into three different application logs corresponding to each class we defined. We then extracted the circuits and their attributes of each class, and eliminated outlier circuits which downloaded only a few cells. From the logs, we created an offline data set and three online data sets and converted them to the Attribute-Relation File Format (ARFF) format for the WEKA processing. We next describe our offline and online datasets

Offline data set. We sampled approximately 200 circuits from the three application traces. Of the 200 circuits, 122 were browsing circuits,[6] 49 were BitTorrent circuits and the remaining 28 were streaming circuits. We then extracted the circuit-level attributes we used for offline classification: the circuit lifetime, the amount of data sent upstream and downstream, and the variance of the cell inter-arrival times of a circuit.

Online data set. From our application traces, we also extracted the cell attributes that were computed online for every cell that the entry node transferred for our clients. Each instance of our online data set is composed mainly of cell-level attributes such as the inter-arrival time of the current cell, its mean and variance. Also, an instance contains some circuit-level attributes such as the EWMA of the count of the cells sent recently on a circuit. We created three different ARFF files using the same data set, but for every file, we used different classes:

- TBS: This file contained sampled attributes from BitTorrent, browsing and streaming circuits. Each data instance was labelled with its respective class. Approximately 60% of the instances were extracted from browsing circuits, 20% were from BitTorrent, and the remaining 20% were from streaming circuits.

[6]Note that the Tor client uses on average 6 circuits per hour for browsing, so 122 circuits is approximately 20 hours' worth of browsing.

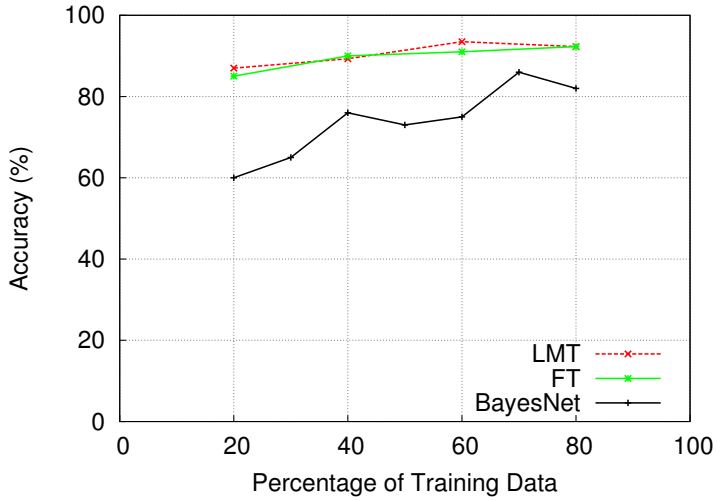


Figure 4.5: Accuracy of different classification algorithms used for the offline dataset when varying the percentage split of training data relative to the whole dataset

- TN: This file contained the same data as TBS, except that browsing and streaming instances were labelled in common as "NotBulk", while BitTorrent instances remained separate as "Bulk".
- BT: This file contained only the subset of browsing and BitTorrent instances from TBS.

4.5.2 Evaluation Metrics

To evaluate our classification algorithms, we utilized two widely used machine-learning metrics, the *accuracy* of the classifier, and the *F-measure*. We use the accuracy to evaluate the overall classification accuracy, whereas the F-measure is used in our evaluation as a per-class evaluation metric.

Accuracy reflects the percentage of the testing instances that have been classified correctly out of all instances:

$$Accuracy = \frac{TP + TN}{N} \tag{4.5}$$

where N is the total number of samples. TN and TP are the true negatives and the true positives of a test, respectively. The F-measure is defined as the harmonic mean of *precision* and *recall*, which are defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{4.6}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.7}$$

where FP and FN are the false positives and false negatives of a test, respectively. Therefore, the F-measure is given by:

$$F\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4.8}$$

Intuitively, a score of 1 for the accuracy or the F-measure indicates a perfect classifier.

4.5.3 Offline Classification

In this section, we present our results for the Bayes Nets, LMT and FT classifiers, which we overviewed in Section 4.4.3, on our offline dataset. Figure 4.5 shows how the accuracy of the classifiers changes as we vary the percentage of the training data relative to the whole dataset. The figure shows that the accuracy of Bayes Nets seems to improve from 60% when training data is only 20% of the offline dataset to above 85% when the training data is around 70% of the offline dataset. The Bayes Net classifier starts to degrade to around 82% as the percentage of training data increases to 80%. This suggests that more data than 70% of the offline dataset can add noise to the classifier. FT and LMT provide better accuracy, and are very similar to each other. With around 60% training data, each classifier provides us with greater than 90% accuracy.

Figure 4.6 summarizes the accuracy of the classifiers when 10-fold cross-validation is used on our offline dataset. In the n-fold cross-validation, the training data is divided into n subsets. One subset is used for testing and the other $n - 1$ subsets are used for training. This process is repeated n times so that in every iteration, a new subset is used for testing. The advantage of this method is that the whole dataset is used for both training and testing, and the n results can be averaged. As seen in the figure, the FT classifier achieves 91% accuracy when 10-fold cross-validation is used.

Figure 4.7 depicts the F-measures of the different classes when different classifiers are used in combination with 10-fold cross-validation. Again, FT provides the highest F-measures for all of the classes. It can also be seen that all three classifiers perform well in identifying a browsing circuit, as their F-measure for the browsing class ranges from 0.83 for Bayes Nets to 0.97 for FT. FT and LMT classifiers also perform very well in identifying BitTorrent circuits. However, they both achieve around 0.71 for the streaming class.

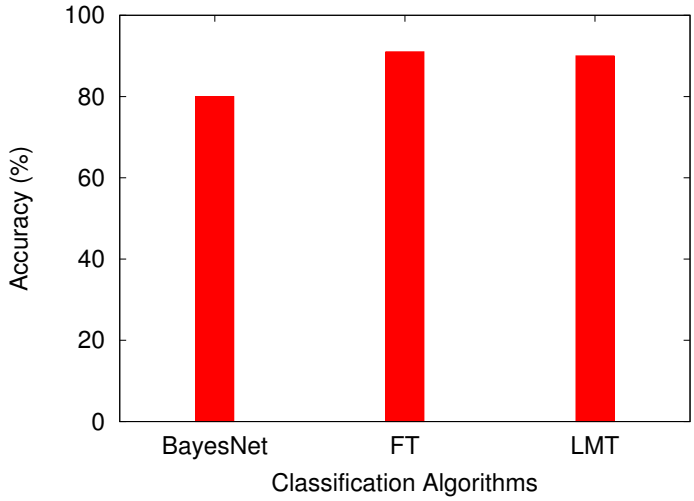


Figure 4.6: Accuracy of different classification algorithms on the offline datasets when 10-fold cross-validation is used

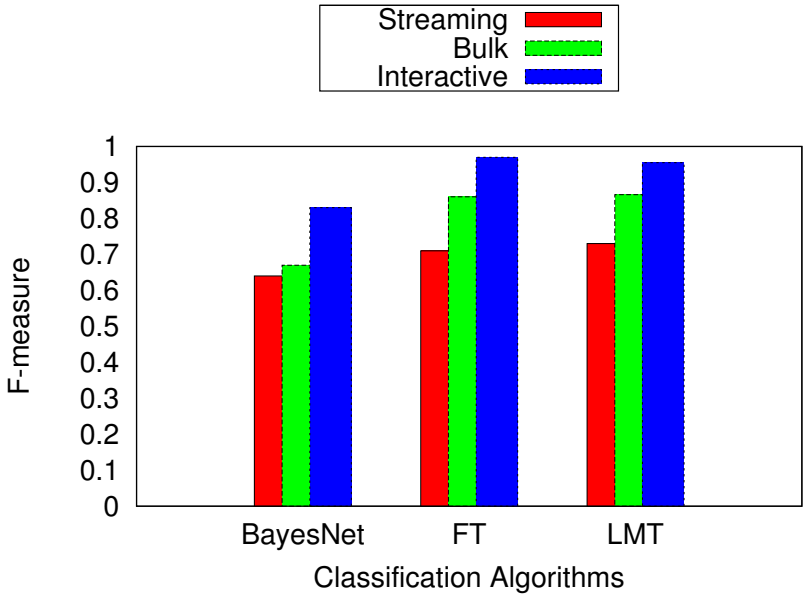


Figure 4.7: F-measure of the different traffic classes when using different classification algorithms with 10-fold cross-validation on the offline dataset

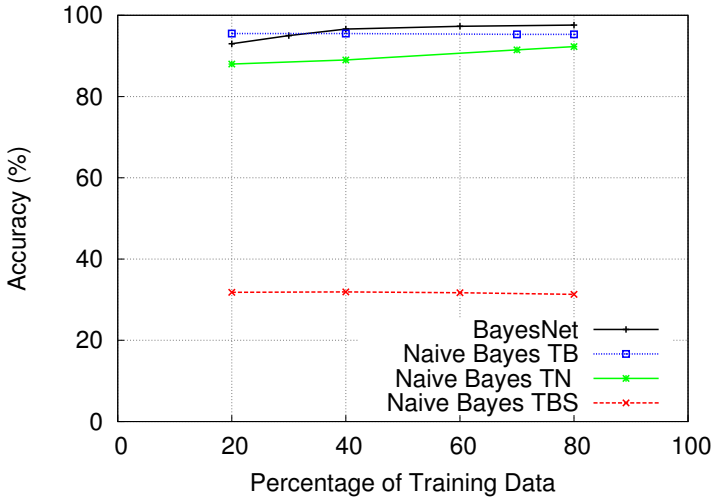


Figure 4.8: Accuracy of different classification algorithms on the online datasets when varying the percentage split of the training data relative to the whole dataset

4.5.4 Online Classification

We next present our classification results for our three online datasets. For the TBS dataset, we used both Bayes Nets and Naïve Bayes. Figure 4.8 shows the classifier accuracy as we vary the percentage of the training subset of TBS. With only 30% training percentage, Bayes Nets classifies most cells with an accuracy of 95%. Naïve Bayes on the other hand provides a very poor accuracy on the TBS dataset which does not exceed 31% regardless of the percentage of the training data. Figure 4.9 shows the accuracy of Bayes Nets and Naïve Bayes classifiers on the TBS dataset when 10-fold cross-validation is performed. Bayes Nets provide a 97.8% accuracy while Naïve Bayes provides a 31% accuracy.

However, the accuracy of Naïve Bayes jumps to 85% if used to classify the TN dataset. This shows that the Naïve Bayes classifier is good in identifying the bulk class, and other non-bulk classes. The Naïve Bayes classifier further shows more accuracy on the TB dataset which reaches 95%. It can be seen from these results that the Naïve Bayes classifier performs best when we eliminate the streaming class training, as it is an excellent classifier in detecting the bulk and the interactive classes. This is evident from the F-measures we obtained for the bulk and interactive classes shown in Figure 4.10. Table 4.1 summarizes a high-level comparison between our threshold, offline, and online classifiers.

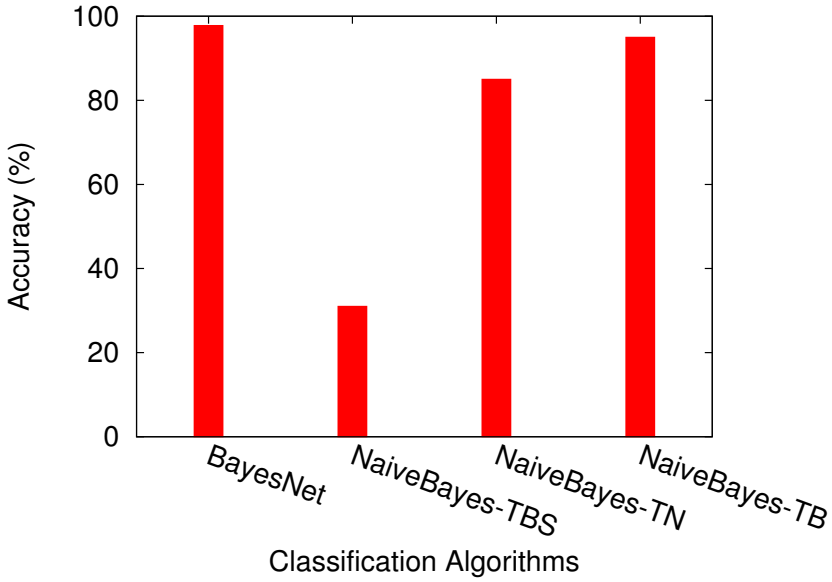


Figure 4.9: Accuracy of different classification algorithms on the online datasets when 10-fold cross-validation is used

Table 4.1: Comparison of threshold, online and offline classification methods

Classification type	*Classification level*	*Accuracy*	*Attribute(s)*
Threshold	Connection-level	90%	Number of circuits per connection
Offline	Circuit-level	91%	Data transferred, CIT variance, and circuit lifetime
Online	Cell-level	97%	CIT statistics and EWMA of cells sent

4.5.5 Live Tor Experiment

In order to show the performance benefits that are made possible using traffic classification, we implemented a Naïve Bayes classifier in the Tor source code.[7] We used the training model we obtained from WEKA to train our classifier. In particular, we trained our classifier to recognize two traffic classes, the interactive and the bulk transfer classes. The reason we chose this classifier is twofold. First, as previously discussed, the largest application that is used on Tor is web browsing, and the second largest is BitTorrent. Therefore, our classifier should be able to identify the largest two classes of traffic that use the Tor network. Secondly, online classification requires classification at the cell level; therefore, it is important to ensure that our classifier is efficient

[7]We used tor-0.2.2.35 for our experiments.

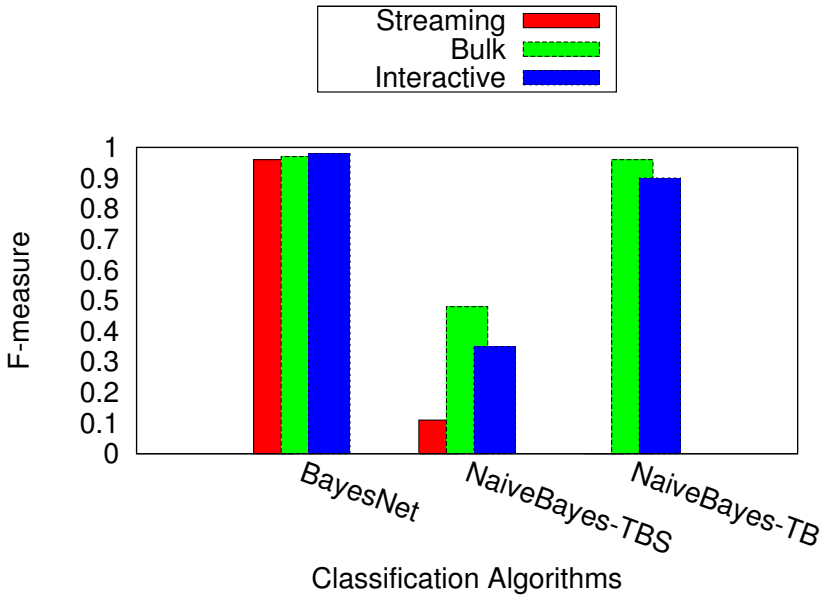


Figure 4.10: F-measure of the different traffic classes when using different classification algorithms with 10-fold cross-validation on the online dataset

and does not add any extra load on the operation of an OR. A simple Naïve Bayes classifier fits those two conditions, and its running costs are minimal.

We deployed a Tor entry node,[8] which runs our classification implementation, on the live Tor network in April 2012. We also implemented the following simple QoS rule.[9] If our OR classifies at least three circuits in one client connection as bulk, then the OR reacts by throttling the client connection to 50 KB/s. The OR decides the class of the circuit after classifying the first 3000 cells that travel downstream in the circuit towards the client. The OR decides that a circuit is bulk if the number of cells classified as bulk is at least two times the number of cells classified as interactive.

Recall from Section 4.5 that we *only* monitor our own traffic, so as to not accidentally deanonymize real Tor users.

We found that our classifier was able to classify bulk and interactive circuits accurately after classifying approximately 2000 and 1000 cells, respectively; however, we chose the 3000 cell

[8] To induce congestion, we capped its bandwidth to 200 KB/s.

[9] Note that our live experiment is a proof-of-concept, rather than a precise throttling rule that should be used in practice. Because we showed in previous sections the possibility to classify Tor circuits accurately to three classes, one has the flexibility to define different QoS techniques to react to different classes.

Table 4.2: Overall and per-class accuracy of live experiments

Overall	*Bulk class*	*Interactive class*
77%	74%	95%

mark in order to have more confidence in our classification decision. Furthermore, although 3000 cells might sound like a large number of cells, we argue that 3000 cells are transferred by bulk applications in a matter of a few seconds.

We ran two Tor clients from the same machine. Our first client is the Vuze BitTorrent client, which acts as the bulk class traffic generator. Its setup remains similar the one described in Section 4.5. The second client, the interactive class traffic generator, runs a curl script in a loop to download a 300 KB file from an external server through Tor using our entry node as its first hop. The client pauses randomly for 3 to 30 seconds and then sends another download request to the external server to start downloading. Note that the behaviour of this interactive client is slightly different from the automated web browser we used to train our classifier because of the absence of the effects of dynamic contents, for example.

Figures 4.11 and 4.12 depict the performance of our interactive class user when the OR runs stock Tor and when it runs DiffTor. As can be seen in Figure 4.11, the download time — the time it takes the Tor client to complete downloading a 300 KB file — is significantly improved. While the download time that the client experiences is 16 seconds at the median without QoS, the median download time is substantially improved to 2.2 seconds when QoS is employed at the OR. Likewise, the time-to-first-byte, the time it takes the client to receive the first chunk of data after issuing a request, is also significantly reduced from 3.5 seconds at the median for the non-QoS case to 0.9 seconds when QoS is used, as shown in Figure 4.12. Table 4.2 shows the overall and per-class accuracy of the live experiments. To summarize our results, the download time is improved by 86% and the time-to-first-byte is improved by 75% at the median for the web browser. As for the BitTorrent client, the download rate was successfully throttled to 50 KB/s as intended by our example QoS rule.

4.6 Discussion

Below we discuss a variety of issues such as the deployability of our scheme on the live Tor network and the effects of human variability on the performance of our techniques. We also express our thoughts on the effects of gaming our classification techniques and on the security

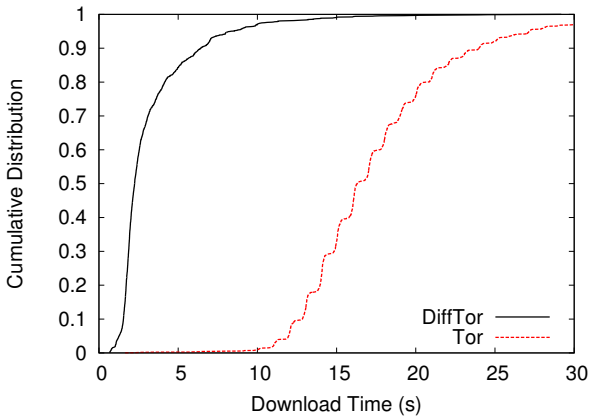


Figure 4.11: Comparison of the 300 KB download times that the web client experiences with and without QoS

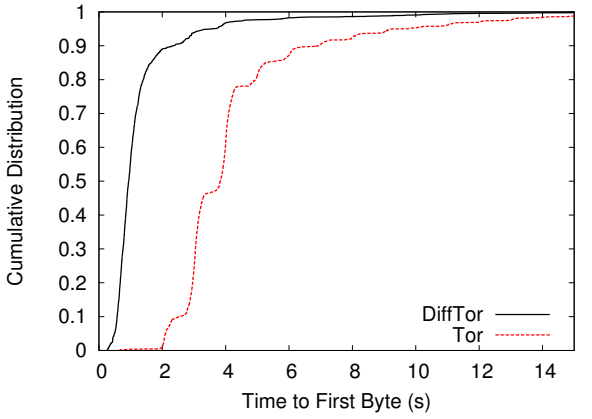


Figure 4.12: Comparison of the time-to-first-byte that the web client experiences with and without QoS

and privacy implications of traffic classification in Tor. Finally, we talk about open issues that we plan to investigate as future work.

4.6.1 Incremental Deployment

Our work is currently applicable to entry guards and is incrementally deployable. Setting QoS parameters can be done locally by OR operators or globally by directory authorities. Although

we have only demonstrated how we can implement a simple QoS rule that throttles a client connection when it detects a bulk transfer at an entry guard, our work extends naturally to middle and exit ORs. While it is infeasible for a middle or an exit OR to throttle connections for the sake of implementing QoS,[10] we can still implement QoS rules for middle or exit routers at the circuit scheduling level (as opposed to the connection level) to reduce the effects of congestion on interactive applications; this could be particularly useful to counter bulk downloaders who run their own (non-throttling) entry nodes. Furthermore, we suspect that QoS can provide even further improvements if combined with fair queuing on all circuits [TS11]. However, maintenance will be sometimes required: as the behaviour of applications change, and with continuous changes to the Internet, the classifier will need to be retrained to maintain its accuracy.

4.6.2 Human Variability

Because we used automated tools to generate web and streaming traffic in our data collection phase, one might wonder if our techniques would generalize to a broad range of users who use applications differently. In our work, we are confident that we addressed this issue in our realistic traffic generation and in our selection of solid attributes. We strived to generate as much realistic traffic as possible by exploiting previous studies that estimate user think times for both streaming and interactive applications, and by using the most popular URLs and keywords. Second, we also chose attributes that are strong enough to prevail even in face of user variability. Recall that our attributes identify burstiness, the variance of burstiness, and the amount of data sent recently. Those are very high-level application signatures that should not be different for different humans because they are influenced by how an application protocol behaves.

For example, even if different users employ different BitTorrent clients, our techniques will still identify the behaviour of the BitTorrent protocol. The BitTorrent protocol will always attempt to create several streams which would create many circuits and result in what looks like a continuous downloading circuit. While there is more variability in the interactive class since it involves human interaction, such variability will only affect the think times observed, but will not change the fact that interactive applications are bursty.

4.6.3 Gaming the Classifier

Because we perform classification at the circuit and cell levels, some users may change the behaviour of their applications to avoid certain QoS measures. For example, if an entry node

[10] Recall that an OR-to-OR connection multiplexes circuits from several users.

throttles bulk transfers, then BitTorrent users might shape their traffic so that it is not classified correctly as bulk. This attempt to disguise one class of traffic's statistical features (e.g., packet sizes and timing) as another traffic class's features, called *traffic morphing* [WCM09], has been proposed as a way to defeat statistical traffic classification techniques. We note that, even when armed with traffic morphing techniques that are informed with prior knowledge about the classification features and the training model, it is often the case that identifying features persist that reveal the true application [DCRS12].

With regard to our classification approaches, there are two scenarios to consider. First, to game the offline classification, in addition to shaping their traffic to look more bursty, BitTorrent users would have to send significantly less traffic upstream and downstream, and they would need to discontinue using the circuit within 10 minutes. We argue that shaping their traffic in this way would still be a win to the Tor network since that would significantly reduce congestion. Second, to game the online classification, if the classifier makes a classification decision periodically, and changes the QoS parameters accordingly, then BitTorrent clients would still need to continuously shape their traffic to look like interactive circuits, which again is a win to the network. However, BitTorrent clients can still use several circuits over the Tor network to achieve the desired aggregate bandwidth. Preventing such cheating users from obtaining an unfair bandwidth share is an area for future work.

4.6.4 Security and Privacy Implications

There are two security implications that traffic classification raises in Tor. First, an exit may be able to reduce the anonymity set of the entry guard using the circuit if it observes changes in the service. The extent of how much information is revealed depends on the QoS parameter defined. For example, if an entry guard throttles only bulk downloads, then the anonymity of bulk download users may be reduced. However, as more entry guards upgrade, it will be more difficult for an exit to guess who the entry guard is.

Second, since guards are directly connected to users, classifying their circuits can reveal private information about the activities of the users and how they use Tor. Since Tor mainly welcomes interactive real-time applications and since the majority of Tor circuits carry browsing traffic, then bulk download users actually deviate from the behaviour of the majority of circuits and thereby endanger their privacy. Actually, it is already well known that the anonymity of BitTorrent users on Tor is questionable for several reasons [Din10], so classification does not really add further damage to the anonymity of this class of users [BMC+11]. Our work is another disincentive for greedy users to deviate from typical network usage.

4.6.5 Future Work

There are still some interesting areas for investigation in our work. First, as the network grows and starts to attract more users, we expect that more applications will emerge in the Tor network and we will therefore need to define more classes of service and investigate more attributes and algorithms. Second, it would be interesting to compare how the different machine learning algorithms trade off cost with accuracy. Another open research question that we would like to answer is how to use the classification techniques we presented in this chapter to infer more information on how users currently use Tor, and the percentage of users of each traffic class. There are currently efforts and proposals to safely collect data on the live network to estimate the number of users [Loe10]. It would be interesting to also log circuit information in a privacy-preserving manner to be able to perform offline classification and understand the current usage of the Tor network. Finally, another area of future work is to investigate different QoS techniques on different traffic classes and perform a whole-network performance evaluation on a Tor network testbed such as ExperimenTor [BSMG11] or Shadow [JH12].

4.7 Related Work

Network traffic classification has been studied intensively in the networking literature [RSSD04, BTA+06, MZ05, ZM05, KCF+08]. Traditional port-based techniques of traffic classification are almost phased out because some applications allocate ports dynamically. Moreover, in addition to their legal and ethical concerns, techniques that are based on deep packet inspection are also ineffective with the use of encryption. Therefore, researchers propose different features extracted from the transport level, packet level and connection level and apply different machine learning techniques to be able to classify traffic to different categories.

Applying the previous solutions to classify traffic in the Tor network is more challenging for two reasons. First, most of the useful features that have been used previously are not available in the context of Tor. Examples include the client and server connection ports used and the packet sizes, which are fixed in Tor. Second, unlike IP network classifications, where packet inter-arrival times is a useful feature, we found that timing features are sometimes fragile in Tor because of the varying circuit and relay characteristics. The problem gets even more challenging as we try to realize real-time traffic classification for Tor.

From the Tor literature, Panchenko *et al.* [PNZE11] describe an attack where the adversary is a local eavesdropper that uses machine learning to infer information from the encrypted traffic sent by a Tor client regarding the web sites that the client visits. The authors first assumed a closed-world model where an attacker trains a classifier with some URLs that the client is known

to visit. Later, the classifier is used to figure out what URLs the client is requesting. They then experiment with an open-world model where the attacker only wants to find out if the client is visiting a restricted website. For both models, the authors show high true positives and low false positives.In our work, we try to classify circuits to different classes depending on the application using the circuit mainly for performance improvement goals. Also, while the features used by Panchenko *et al.* are very useful, we cannot benefit from the packet sizes for online classification at the entry guards, for example, as Tor uses fixed-sized cells.

As described in Section 2.4.1, other related work in the Tor literature was introduced by Moore *et al.* [MWS11] and Jansen *et al.* [JSH12]. From a high level, both proposals are similar in the sense that they both attempt to improve the performance of interactive applications in Tor by throttling bulk transfers. This is done by throttling all connections between the client and the entry guard using Tor's already-implemented token-bucket system. The approach of Moore *et al.* is to apply a certain limit that would slightly affect web browsing users, but would greatly slow down bulk transfers. The only way for clients to avoid the throttling is by donating bandwidth to the Tor network by running as relays. This incentives-based approach and other similar approaches such as BRAIDS [JHK10], which also provides differentiated services, and the gold star scheme [NDW10], are impractical as it is difficult to expect that Tor clients would run relays, and applying it may result in discouraging clients from using the network.

Jansen *et al.* propose and investigate three algorithms that adjust or throttle the connection between an entry guard and a client. Their most effective algorithm, nicknamed the threshold algorithm, maintains an EWMA value of the cells sent on a client connection. This value is used to sort circuits from loudest to quietest, with the goal of throttling a loudest threshold of all connections.

We note that the problem with this approach is that it would unnecessarily throttle time-sensitive streaming applications and because the threshold is high,[11] it may also throttle interactive circuits. Another problem with the threshold algorithm is that it is based only on the EWMA of the cells sent recently. Our experiments revealed that this value alone is sometimes unreliable as interactive circuits sometimes have large values of EWMA while they are downloading web pages.

The advantage of DiffTor over previous approaches is that applying QoS is more flexible since we define different classes of service. For instance, an entry guard can choose to give a defined small percentage of its bandwidth to BitTorrent, and assign more bandwidth to the other two classes. However, when the node is currently not serving streaming or web browsing, it can offer more bandwidth to the bulk transfer circuit, thereby not wasting bandwidth.

[11] The authors report that a threshold of 90% yields the best results.

4.8 Conclusion

Motivated by the crucial need to improve the performance of the Tor network, we propose a machine-learning-based approach to provide differentiated services. We recognize that the current main traffic classes that use the Tor network have different constraints that can be addressed by defining appropriate QoS measures. Based on our study of traffic logs (of our own network usage) we obtained from the live Tor network, we derive useful attributes and use them in combination with well-known classification algorithms to classify our traffic logs with a very high accuracy. Furthermore, we implement our classifier in Tor and define a simple QoS rule to test our approach on the live network. Our results show high classification accuracy that results in significant improvements for the experience of interactive Tor users.

Chapter 5

PCTCP: Per-Circuit TCP-over-IPsec Transport for Anonymous Communication Overlay Networks

Recently, there have been several research efforts to design a transport layer that meets the security requirements of anonymous communications while maximizing the network performance experienced by users. In this chapter, we argue that existing proposals suffer from several performance and deployment issues and we introduce PCTCP, a novel anonymous communication transport design for overlay networks that addresses the shortcomings of the previous proposals. In PCTCP, every circuit is assigned a separate kernel-level TCP connection that is protected by IPsec, the standard security layer for IP.

To evaluate our work, we focus on the Tor network. We believe the current transport layer design of Tor, in which several circuits are multiplexed in a single TCP connection between any pair of routers, is a key contributor to Tor's performance issues.

We implemented, experimentally evaluated, and confirmed the potential gains provided by PCTCP in an isolated testbed and on the live Tor network. We ascertained that significant performance benefits can be obtained using our approach for web clients, while maintaining the same level of anonymity provided by the network today. Our live network experimental evaluation of PCTCP shows improvements of more than 74% for response times and more than 76% for download times compared to Tor. Finally, PCTCP only requires minimal changes to Tor and is easily deployable, as it does not require all routers on a circuit to upgrade.

5.1 Introduction

Recall that previous research [RG09] has shown that the weaknesses in the design of Tor's transport layer are capable of impinging on its performance. As a design solution, Reardon and Goldberg proposed TCP-over-DTLS (described in Section 2.4.1), where every circuit gets a separate user-level TCP connection, and DTLS is used for encrypting and securing the communication between routers. Unfortunately, TCP-over-DTLS faces the following design drawbacks:

- **Performance:** User-level implementations of TCP provide significantly lower performance than their kernel-level counterparts in terms of throughput and consume substantially more CPU cycles [EM95, BDHR95], a scarce resource in Tor. Such heavy costs might render any performance benefits moot if a user-level TCP scheme is deployed at a wide scale.
- **Deployability:** First, the unavailability of a reliable user-level TCP stack with a license that is compatible with Tor is a major obstacle facing TCP-over-DTLS.[1]Second, for any pair of routers to use TCP-over-DTLS, both routers need to upgrade their transport design.

Our Approach. In this chapter, we seek to enhance the performance and usability of the Tor network for interactive application users. We tackle the performance problem in Tor at its roots, and focus on fixing the weaknesses in Tor's transport design. We propose PCTCP, a new transport design for Tor in which a separate kernel-level TCP connection is dedicated to every circuit. To protect and secure communication between routers, we use IPsec, the standard security layer for IP. Our design significantly improves the performance of Tor while maintaining its threat model. Additionally, PCTCP requires only minimal changes to the software. Our design combines the advantages of the previous TCP-over-DTLS proposal, while avoiding its deployment and performance shortcomings, inherent from using a user-level TCP stack. Furthermore, PCTCP does not require all routers on the circuit to upgrade, except for enabling IPsec communication for a pair of routers that wish to use DiffTor. Our design has a significantly easier road to deployment.

Contributions. This is the first work that implements a new transport design, for anonymous communication systems in general and for the Tor anonymity network in particular, and evaluates it with realistic large-scale experiments, as well as live network experiments. In designing and implementing PCTCP, we offer the following contributions:

- We propose and implement PCTCP, a novel transport design for anonymous communication systems in general and for Tor in particular that avoids the deployability and performance drawbacks of previous designs.

[1]Reardon and Goldberg used the Daytona TCP stack for their implementation and measurements. Unfortunately, Daytona cannot be used for the Tor network due to its unavailability for open-source projects.

- We evaluate our design by performing a series of large-scale emulation experiments on a topology that closely approximates the performance of the live Tor network. Our results show significant performance benefits for the download and response times of web clients.
- We carry out experiments on the live Tor network to validate our results. Again, our results show significant reductions in delays observed. Our response times are improved by 27% at the median and by 74% at the 75th percentile. Moreover, download times are improved by 55% at the median and by 76% at the 75th percentile.
- Our simple, yet powerful and effective, approach is incrementally deployable, as our changes, except for enabling IPsec communication between any pair of routers using PCTCP, are local to individual routers and do not affect their operation with other routers.

The rest of the chapter is structured as follows. We provide the reader with the necessary background on IPsec in Section 5.2 and compare our work to previous work in Section 5.3. Then, we elaborate on our design in Section 5.4 and evaluate it in Section 5.5. Finally, we discuss some open issues regarding our design and experiments in Section 5.6 and conclude in Section 5.7.

5.2 IPsec

IP security (IPsec) [KA98] is a collection of standards that provides security at the network (IP) layer. It defines several protocols that enable authenticating and/or encrypting IP data packets. It consists of mainly two sub-protocols: *Authentication Header* (AH) and *Encapsulating Security Payload* (ESP). We next briefly describe each sub-protocol and their modes of operation.

The AH protocol allows two communicating points to authenticate, and protect the integrity of the data they exchange. Although the AH protocol guards against spoofing and replay attacks, it does not encrypt the data traveling between the two ends, so an eavesdropper can view the contents of the data packets.

The ESP protocol, on the other hand, enables both authentication and encryption (or either), which provides confidentiality of the transferred data. The two communicating ends need to have secret keys in order to decrypt the packets. IPsec provides a variety of key-exchange and authentication algorithms.

For both protocols, there are two modes of IPsec operation: either the *transport* or the *tunnel* mode. Transport mode is used to secure the connection, consisting of the traffic from different applications, between two hosts. The payload of the IP packet, which typically contains TCP or

UDP data, is encrypted or authenticated and an ESP or an AH header is added to the packet. The original IP header also remains in the packet.

Tunnel mode, on the other hand, secures not only host-to-host communication, but it also can be used to protect communication between subnets to subnets or hosts to subnets. In this mode, the whole IP packet is encrypted or authenticated and a new IP header is added to the encrypted packet in addition to the AH or ESP header. Using the ESP protocol in tunnel mode provides the strongest security for communication at the expense of a few extra bytes per packet as an overhead. However, when only host-to-host communication is required, ESP protocol in transport mode suffices. In the next section, we present previous work on anonymous communication transport design for Tor. After that, we introduce our proposed anonymous communication transport for Tor and how we use IPsec to secure communication between Tor ORs.

5.3 Related Work

New transport designs for Tor have been investigated and considered by several previous proposals [Vie08, RG09, TS12]; Murdoch [Mur11] provides a summary and compares all these previous possible transport designs. He categorizes the available designs into three different architectures: hop-by-hop reliability, initiator-to-exit reliability or initiator-to-server reliability. Although Murdoch does not experimentally evaluate these design choices, he expects that a hop-by-hop reliability approach will be the most promising approach. Next, we summarize the first two design categories and contrast them with our design. For more details on the initiator-to-server design architecture, we refer the reader to Freedom [BSG00] and Murdoch's summary [Mur11].

TCP-over-DTLS [RG09] is an example of the hop-by-hop reliability design, which is also the same design approach we adopt in PCTCP. The TCP-over-DTLS proposal advocates for using a user-level TCP connection to manage every user circuit over DTLS—the datagram alternative to TLS—to provide confidentiality and authenticity of Tor's traffic. Since every circuit is managed by its own TCP connection, every circuit is guaranteed reliability and in-order delivery of cells. Furthermore, congestion control is performed at the circuit level, which solves the cross-circuit interference problem.

Several differences separate PCTCP from TCP-over-DTLS. First, PCTCP uses mature IPsec protocols to hide TCP/IP header information, whereas TCP-over-DTLS uses the relatively rare DTLS for the same purpose. Also, TCP-over-DTLS introduces deployment and performance issues that hinder its adoption (as highlighted in Section 5.1). PCTCP avoids these problems by using the kernel-level TCP stack, and by having an easier path to deployment. Second, while initial experiments on TCP-over-DTLS performed on a localhost private Tor network showed

slightly less degraded latency results, as compared to Tor, when packet drop rates increased, there is still a need for further realistic large-scale experiments in order to obtain conclusive results of the potential benefits. With the lack of such experiments in previous work, it is difficult to compare TCP-over-DTLS and PCTCP in terms of performance gains.

UDP-OR [Vie08] is an example of an initiator-to-exit reliability design. In this design, an OP and the exit OR of the circuit maintain a TCP connection, while intermediate ORs communicate using UDP, an unreliable transport protocol. While this design significantly simplifies the operations of the intermediate routers, it still suffers from several problems. The first problem is that since hop-by-hop communication is unreliable, there will be a need to change the cryptographic protocols that are implemented in Tor as the current circuit encryption scheme depends on in-order delivery of cells. Another problem is that this design uses the OP's host TCP stack, rather than a user-level one, which opens the door for OS fingerprinting attacks [KBC05]. Second, since a circuit's round trip time is large, it would take the TCP endpoints a significant amount of time before congestion is triggered. Also, with the high variability of circuit performance in Tor, a non-trivial amount of tuning for TCP parameters, including congestion timers, thresholds and windows, may be required for the TCP endpoints; see Section 5.4 for more details.

Torchestra [GH12] was recently proposed to enhance the performance of interactive application users of Tor. In that proposal, two TCP connections are used for OR-to-OR communication. One TCP connection is dedicated for light circuits and another is dedicated for heavy circuits. An EWMA of the number of cells sent on a circuit, originally proposed by Tang and Goldberg [TG10], is used to classify circuits into light and heavy categories. Our work in Chapter 4 suggested that this metric alone is not enough to distinguish circuits.[2] Also, Torchestra has not been examined using large-scale experimentations to understand the system-level effects of utilizing it. Finally, to benefit from Torchestra, all ORs on the circuits need to upgrade, as two TCP connections, as well as new command cell types, are needed between every pair of ORs in a circuit.

Tschorsch *et al.* [TS12] consider the impact of several proposed transport designs for Tor on throughput, packet loss, delay and fairness. For their analysis, the authors use a TCP performance model proposed by Padhye *et al.* [PFTK98]. They examine the performance of several proposed transport designs for Tor using a discrete-event simulator, and conclude that they expect that a joint congestion control that detects loss rates and congestion for all circuits traversing an overlay node would be a good direction. The authors ruled out the use of parallel TCP connections, such as in PCTCP, as a design option, as more connections traversing a bottleneck may result in higher packet losses, which reduces throughput. We argue that packet losses mainly occur for the connections carrying bulk traffic, as they send significantly more data than connections

[2]Unfortunately, the classification accuracy was not discussed in Torchestra [GH12].

carrying interactive applications. We also demonstrate through comprehensive emulation and live-network experiments that our approach is effective.

5.4 Proposed Transport

Before embarking on the description of PCTCP, we first ask ourselves, why not adopt and implement an end-to-end TCP approach, which has been proposed as a possible transport design for Tor. We first start by explaining why we avoided such approach, and then we elaborate on our design.

5.4.1 Why not end-to-end TCP?

One transport design that has received some positive speculation in the Tor research community is the end-to-end TCP design. This design is inspired by many previous proposals [BSG00, Bro02, Vie08]. The basic idea of this design is that a TCP connection is maintained by the two ends of the circuit. In the context of Tor, one end is the client and the other end can be the exit OR or the destination server. Communication between intermediate ORs is carried out using a datagram protocol, such as UDP. We next point out some weaknesses in this design choice.

Tuning Parameters TCP is a reliable transport. If a packet gets dropped or lost due to congestion or routing problems in the underlying IP network, TCP's congestion control algorithm is triggered and the sender retransmits the lost packet. Also, TCP ensures that the Tor process, residing at the application layer, receives data in the order they were sent. This functionality significantly simplifies the task of data processing for Tor. By contrast, a datagram protocol like UDP, or its secure DTLS alternative, do not implement reliability or in-order delivery.

In the end-to-end TCP design for Tor, it is assumed that reliable in-order delivery is maintained only by the end points. There are several shortcomings with this design that might worsen the experience of Tor users. The biggest challenge is how to best tune the TCP parameters to yield a reasonable performance for Tor. TCP relies on duplicate acknowledgement packets sent by the receiver to detect congestion which signals that several out of order packets have been received at the destination. Moreover, TCP also relies on retransmission timers at the sender to detect loss of packets.

Typically, retransmission timers should be equal to the round-trip-time (RTT) between a source and a destination. In a network like Tor, where the RTT of circuits can be several seconds long, it can be easily seen that a client would detect congestion very late. Of course, the client

can set a smaller retransmission timer to detect congestion faster; however, one should be careful not to send redundant packets too quickly, as this might cause even further congestion. Striking a good balance between how fast we want to detect congestion and how careful we should be before we decide we are experiencing congestion is a very difficult problem. Also, considering the timing characteristics of Tor circuits, which are notorious for their highly variable performance, one soon realizes that an end-to-end TCP solution for Tor is unwise.

Interoperability and Anonymity. An important aspect of any new transport design for Tor is to ensure that it can be smoothly integrated to work with the existing Tor network infrastructure without disrupting the operation of the network and its users. Recall that Tor today currently has thousands of ORs and hundreds of thousands of users. The network has not experienced significant downtime since its deployment in 2003. Using a drastically different transport design such as end-to-end TCP would require the network to pause its operation while ORs and users update. As a workaround, it might be possible for ORs upgraded with end-to-end TCP to coexist with unmodified ORs; however, this might open the door for fingerprinting or partitioning attacks. For example, an upgraded malicious exit can reduce the anonymity set of the entry guard used on a circuit from the set of all entry guards in the network to the smaller set of upgraded entry guards. Therefore, one shortcoming of upgrading to an end-to-end TCP design is possibly hindering the anonymity provided by the network.

Cryptographic Protocols. An inherent consequence of allowing an unreliable transport is for the Tor process to expect lost packets. Since Tor uses the Advanced Encryption Standard (AES) in counter mode for encrypting and decrypting cells at ORs, lost or dropped cells will cause subsequent cells to be unrecognized. Therefore, adopting an end-to-end TCP approach requires changing the cryptographic protocols that are currently used in Tor; this is another obstacle facing such a design.

5.4.2 PCTCP

The aim of this work is to address the shortcomings of the transport design in Tor. In particular, our goal is to reduce the impact of the cross-circuit interference problem which hinders the experience of interactive application users. Based on our discussion in Section 5.4.1, we believe that reliability should be maintained on a per-hop basis for Tor circuits. Therefore, in this work, we advocate for maintaining TCP connections between each adjacent pair of ORs that comprise a circuit. In particular, we propose two key design changes to Tor's transport.

Kernel-mode per-circuit TCP

We propose using a separate kernel-mode TCP connection for each circuit for Tor. Our design is similar to the TCP-over-DTLS design that was introduced by Reardon and Goldberg in the sense that reliable in-order delivery of data is implemented between every two communicating ORs. Also, both designs ensure that congestion control is performed at the circuit granularity. The elimination of connection-sharing among circuits ensures that we isolate the effects of loud circuits on the quiet ones. A cell dropped or lost from one circuit will only affect that particular circuit.

However, one key difference between PCTCP and TCP-over-DTLS is that for circuit management, PCTCP uses kernel-mode TCP connections for every circuit, while TCP-over-DTLS uses a user-space TCP implementation. The lack of availability of a reliable open-source user-level TCP stack whose license is compatible with that of Tor hinders the deployability of the TCP-over-DTLS solution. Furthermore, PCTCP uses IPsec to protect the communication between ORs whereas TCP-over-DTLS uses DTLS. One issue that is inherent from using DTLS is that it is rarely used today on the Internet. IPsec, on the other hand, is increasingly common, as it is utilized in many implementations of Virtual Private Network (VPNs) [SSGC05]. Consequently, the rarity of DTLS makes it easier to be blocked by censors without fearing side effects. Blocking IPsec would be more problematic, as blocking it may interrupt the operation of legitimate businesses and organizations.

We next describe how we modify the behaviour of Tor to support PCTCP. Recall that during the circuit construction process, every time an OP attempts to extend the circuit by one more hop, it sends an *extend* command cell to the current last OR on the partially constructed circuit. When an OR X_i receives an *extend* cell to another OR X_j, X_i checks if it has a current TCP connection with X_j. If a connection exists, X_i uses that connection to send the *create* cell; otherwise, it creates a new TCP connection to X_j before a *create* cell is sent.

In PCTCP, when an OR, X_i receives an *extend* command cell to X_j, PCTCP always establishes a new TCP connection from X_i to X_j. This means that in PCTCP, we maintain the same queueing architecture of Tor, except that our design eliminates the contention that occurs among circuits when they share the same connection output buffer, as each circuit queue is mapped to a single output and a single input connection buffer. When a circuit is torn down, its corresponding TCP connections are closed.

Figure 5.1 visualizes a design comparison between Tor and PCTCP. As can be seen in the figure, between an OP and an OR, PCTCP, like Tor, maintains a single TCP connection, which can multiplex several circuits from the same user. However, PCTCP dedicates a separate TCP connection for every individual circuit between any two ORs.

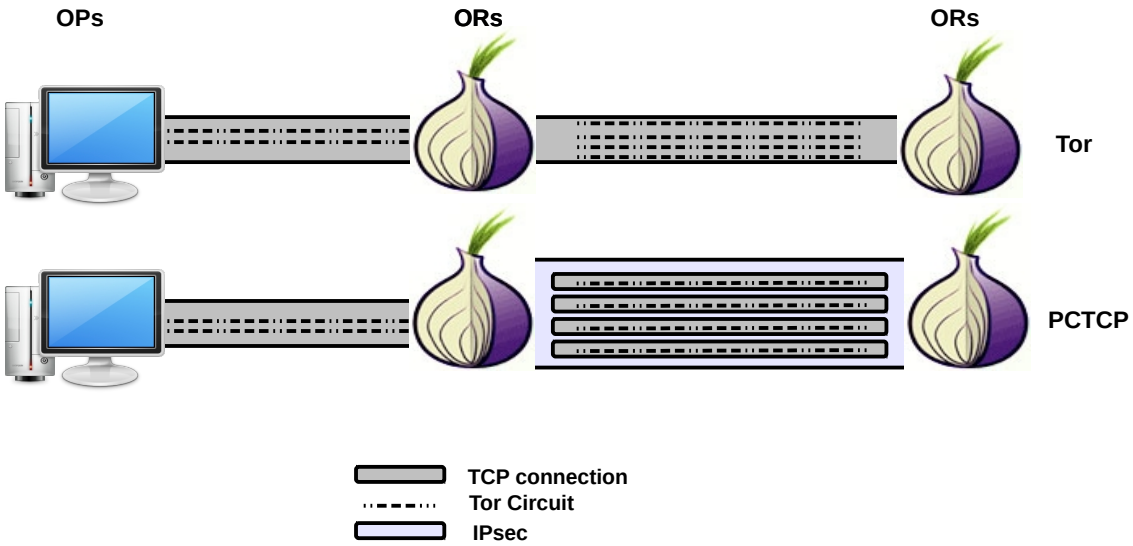


Figure 5.1: Design comparison between Tor and PCTCP. The upper figure shows the current transport design of Tor. An OP maintains a single TCP connection with its entry guard, which also maintains a single TCP connection to the next OR on the circuit. Each TCP connection multiplexes several circuits depicted by the dashed lines. The lower figure shows the design of PCTCP. As before, only a single TCP connection is used between the client and the entry guard; this connection multiplexes all the client's circuits. For OR-to-OR communication, however, several TCP connections, one for each circuit, are created and protected by IPsec.

This design has the advantage that it does not require clients to upgrade, as each client in our design continues to maintain a single TCP connection with each of its entry guards. Moreover, the modifications proposed in PCTCP are only local to each OR. This means that not all ORs in the circuit need to upgrade to benefit from PCTCP. For example, if the middle and exit ORs are the only ORs upgraded with PCTCP on a circuit, that pair of ORs will use PCTCP for their communication even if the entry guard is not upgraded. Nevertheless, we believe that more performance gains can be obtained when more ORs on the circuit upgrade.

Replace TLS with IPsec

One issue that arises with our design so far is that it allows an adversary monitoring a relay to easily count the total number of circuits that are currently serviced by the monitored relay (since each TCP connection corresponds to a circuit). Furthermore, as we have shown in Chapter 4, the

adversary can perform traffic analysis to infer the activity of each circuit. While it is not clear how this extra information can be beneficial for a non-global adversary,[3] there is no doubt that such design reduces the overall anonymity of the system and its users. To alleviate this problem, we propose using the ESP protocol of IPSec in transport mode to encrypt and protect the traffic between the ORs using PCTCP. Since IPsec can encrypt the IP packet payload, TCP connection ports will be encrypted and hidden from an eavesdropper. This makes it more difficult for an adversary to perform traffic analysis on TCP connections between routers. Figure 5.2 compares the format of PCTCP and Tor data packet headers.

Using ESP makes the TLS encryption redundant for PCTCP for OR-to-OR communication, as ESP can provide the hop-by-hop authenticity and data confidentiality that is currently provided by TLS in Tor. Furthermore, like TLS, ESP provides perfect forward secrecy for the data on connections, and prevents an attacker from modifying data. For two ORs to authenticate each other, they can use a certificate-based authentication method that is provided by IPsec. Since ORs issue a long-term identity key that they use to sign their descriptors, they can use the same identity key to sign their IPsec certificates.

Alternatively, ORs can use a public-key authentication approach. An OR could publish its IPsec public key with its signed descriptor to the directory authorities. Then, when other ORs download the descriptors, they can find each other's public keys and use them to start the IPsec connections. Communication between ORs and directory authorities or OPs can continue to use the traditional TLS connections that are used in Tor today.

Ideally, a user-mode IPsec implementation integrated with Tor would be the best option. First, OR operators would not have to deal with the details of setting up IPsec. Second, for user-mode IPsec to operate, superuser privileges are not needed. However, with the lack of an available user-space IPsec implementation, we are only left with the kernel-mode IPsec option. Luckily, installing IPsec is a one-time operation which typically should not require periodic maintenance.

5.5 Experiments

To evaluate the performance benefits possible with PCTCP, we have implemented our proposed transport in a stable release (0.2.2.39) of the Tor source code. Our implementation can be easily turned on or off using a configuration option for any OR. We performed a series of large-scale experiments on an isolated testbed. We also performed small-scale experiments on the live Tor network. As evaluation metrics, we again use the download time and the time-to-first-byte.

[3] Recall that the threat model of Tor assumes an active local adversary.

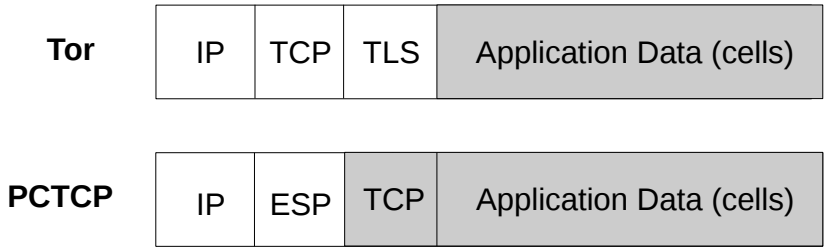


Figure 5.2: Packet headers for current Tor and for PCTCP. The grey shaded area depicts the encrypted part of the packet. The upper figure shows the design of the Tor packets at the network (IP) layer. TLS is used to encrypt the TCP payload, but not the TCP header. The lower figure depicts the packet format when PCTCP is used. The whole IP payload, which contains the TCP segment, is encrypted. An ESP header is added between the encrypted data and the IP header.

5.5.1 Large-scale experiments using a realistic network topology

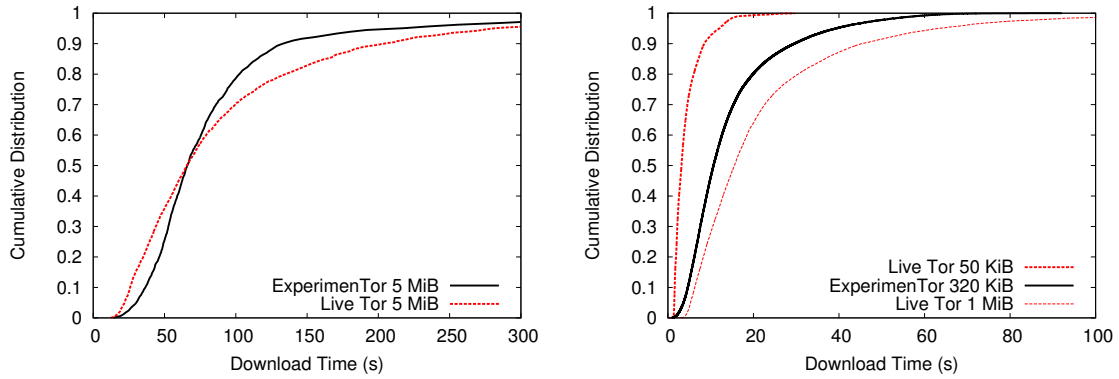
Emulation Tools. In order to understand the system-level effects of our proposed transport, we use ExperimenTor. In our experiments, we use the network and Tor topology models that were recently proposed by Jansen *et al.* [JBHD12] in order to accurately produce a scaled-down Tor network that that closely approximates the performance of the live network.

Underlying Network Topology. We use the network topology that was produced and published[4] by Jansen *et al.* in an effort to facilitate methodically modeling the Tor network for ExperimenTor and Shadow [JH12]. Briefly, the authors form a complete network graph consisting of vertices that correspond to different locations (countries, American states and Canadian provinces) with upstream, downstream and packet loss properties that they obtained from the Ookla Net Index dataset [Oo]. All the vertices are connected by edges with approximated latency,[5] jitter, and packet loss properties.

Overlay Tor Topology. We follow the footsteps of Jansen *et al.* and create a scaled-down topology that consists of 500 Tor clients (OPs), 50 Tor ORs, and 50 HTTP servers. Of the 50 ORs, 5 work as directory authorities. Our ORs are assigned bandwidth values that are sampled

[4]The model files are available for download from the authors' websites (http://www.mit.edu/~ke23793/misc/tormodel_exptor.tar.gz).

[5]The authors use iPlane [iPl] RTTs to approximate latency.



(a) (b)

Figure 5.3: Comparison between the performance of torperf (Live Tor), and our scaled-down testbed Tor network (ExperimenTor). Figure 5.3(a) depicts the download time measurements for downloading 5 MiB files for torperf and ExperimenTor. The results obtained from ExperimenTor closely approximate the distribution of the live network (torperf) and the medians intersect at 65 seconds. Figure 5.3(b) shows how the download time distribution of 320 KiB, obtained from ExperimenTor for stock Tor, fits between torperf's download time distributions for 50 KiB and 1 MiB. These measurements suggest that our experimental setup accurately reflects the performance of the live Tor network.

from the bandwidth distribution of the live Tor network ORs. We create two client types: web clients and bulk clients. Our client model is based on a previous study of the exit Tor traffic by McCoy *et al.* [MBG+08]. During our experiments, our web clients continuously fetch fixed-sized 320 KiB files, and pause randomly for 1 to 30 seconds between fetches.[6] Our bulk clients continuously download 5 MiB files without pausing. Finally, as recommended by Jansen *et al.*, our web-client-to-bulk-client ratio is 19:1.[7]

Model Accuracy. Before we present our results, we first compare the performance of our stock Tor bulk and web clients, which we obtained from our testbed, to the performance of the live Tor network published by the Tor metrics portal [To12b]. This comparison step was also carried out by Jansen *et al*. The purpose of this step is to confirm that our testbed measurements can

[6]While it is possible to use more realistic user think time distributions, we observe such distributions would result in a much lower load as they tend to be long-tailed.

[7]Note that the overlay Tor topology described here is different from the one described in Section 3.4.2, which used 20 Tor ORs and 400 clients. Also, in the topology described in Section 3.4.2, bulk clients model streaming users that pause randomly between 1 to 5 minutes between downloads, whereas the above bulk clients perform continuous downloads to act as traffic generators.

indeed approximate the measurements taken from the live network, even though our network is significantly scaled down.

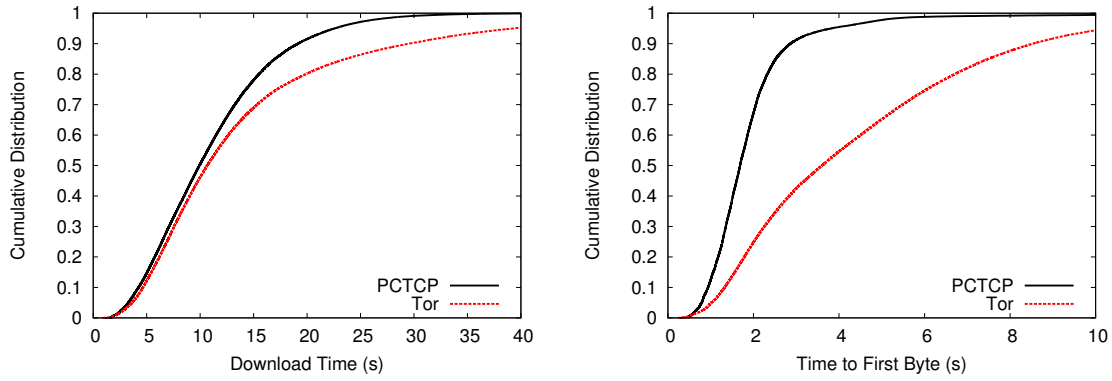
Figure 5.3(a) compares the distribution of the download times of our testbed bulk downloaders and those measured by torperf, a tool that measures download performance on the live Tor network. As can be seen in the figure, the two distributions display comparable performance and they indeed intersect at the median. That is, 50% of the 5 MiB downloads take 65 seconds or less on the live network, and the same is true on our testbed. Figure 5.3(b) compares the results of our 320 KiB downloads and torperf's 50 KiB, and 1 MiB downloads.[8] As expected, the distribution of download times for our web clients fits between the distributions of download times between torperf's 1 MiB and 50 KiB file downloads.

Results. Now that we have verified that our Tor model closely approximates the performance of the Tor network, we next shift attention to our results. Figure 5.4(a) compares the download time observed by web clients when stock Tor and PCTCP are used. The figure shows similar download times for the fastest 50% downloads as these downloads are most likely performed when less congested ORs are used for circuits. However, the figure shows significant improvement for the slowest 50% of the downloads, especially for the fourth quartile of the download times. For example, the download times for Tor range from 17 to 90 seconds, whereas for PCTCP, the download times range from 14 to 56 seconds.

Figure 5.4(b) shows significant time-to-first-byte improvements when PCTCP is used, as compared to Tor. At the median, it takes Tor clients 3.6 seconds before the browser starts changing for them, whereas PCTCP clients only wait for 1.6 seconds, which is a 55% improvement. For the 75th percentile response times, the time-to-first-byte is only 2 seconds for PCTCP users, whereas Tor clients experience delays of up to 6 seconds. This increases the observed improvements to 66%.

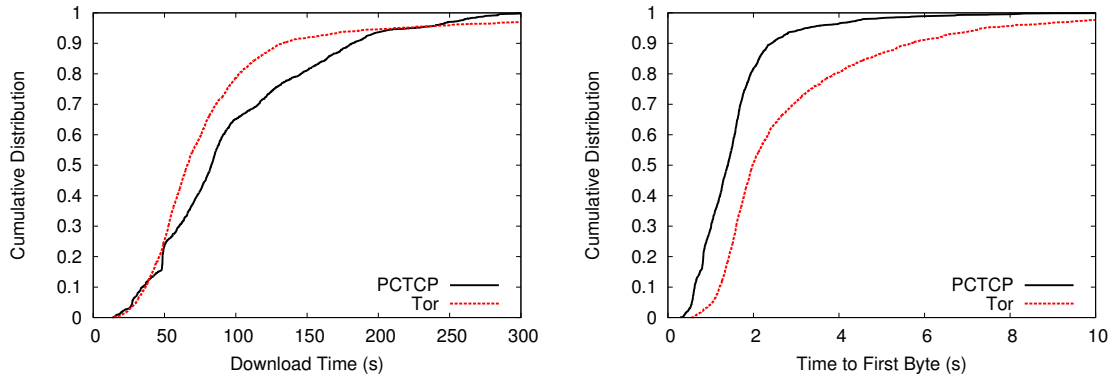
We observe in Figure 5.5(a) that download times for bulk clients are actually degraded when PCTCP is used. For example, the median download time for stock Tor is 65 seconds, whereas for PCTCP, the median download time is approximately 82 seconds. For 60% of the download times, the degradation is roughly 26%. With PCTCP, heavy circuits might observe more delays because such circuits are expected to drop more cells, and their respective TCP connections would back off more frequently as a result of the separate TCP congestion control. This is also consistent with the observations of Tschorsch *et al.* [TS12] that we summarized in Section 5.3. However, we believe that performance improvements can be observed even for bulk clients if more bandwidth was available, as we show in live experiment 2 and in the experiments in Section 5.5.2, below. The time-to-first-byte results are significantly improved for the bulk downloaders, as can be seen in Figure 5.5(b). This suggests that congestion is vastly reduced in the network.

[8] Torperf only maintains the results of 5 MiB, 1 MiB and 50 KiB file downloads.



(a) (b)

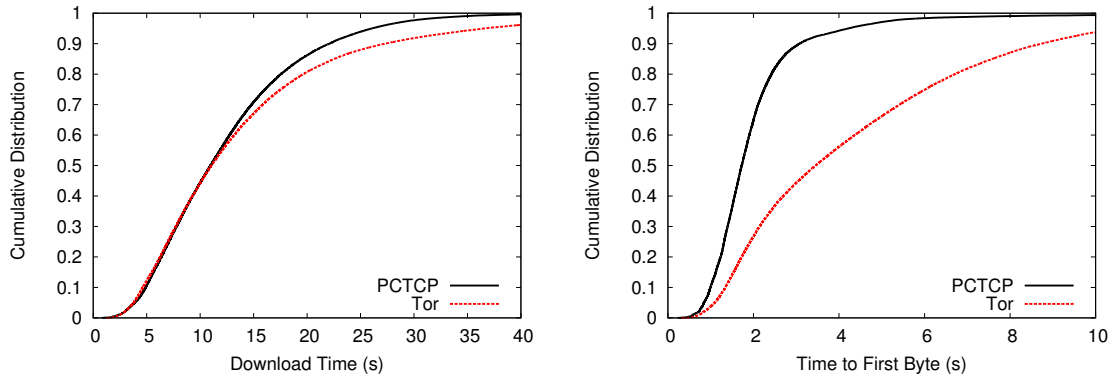
Figure 5.4: Performance of the web clients in the large-scale experiment. Figure 5.4(a) shows the download time distributions using PCTCP and stock Tor, and demonstrates a substantial improvement for the long tail. Figure 5.4(b) shows the time-to-first-byte results using PCTCP and Tor. We see significant improvements using PCTCP, as compared to Tor.



(a) (b)

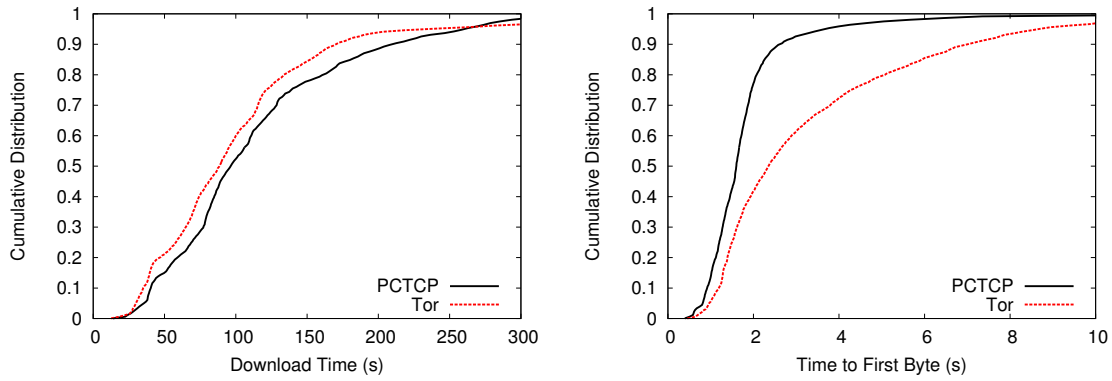
Figure 5.5: Performance of the bulk clients in the large-scale experiment. Figure 5.5(a) shows the download time distributions using PCTCP and stock Tor, and shows a 26% degradation for 60% of the downloads when PCTCP is used. Figure 5.5(b) shows the time-to-first-byte results using PCTCP and Tor. We again see significant improvements using PCTCP, as compared to Tor.

We have also repeated the same large-scale experiments using a 9:1 web-to-bulk-client ratio in order to test PCTCP under exaggerated congestion loads, and our results consistently showed



(a) (b)

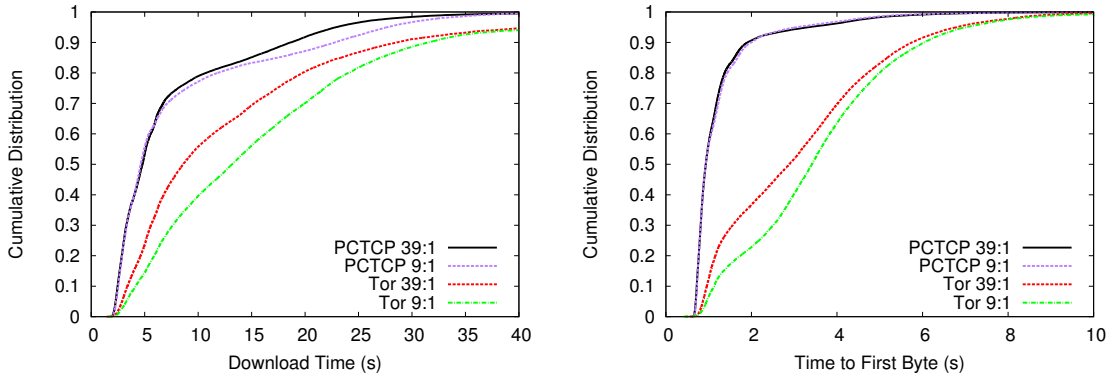
Figure 5.6: Performance of the web clients in a network of 500 clients and 50 routers with a 9:1 web-to-bulk client ratio. Compare to Figure 5.4.



(a) (b)

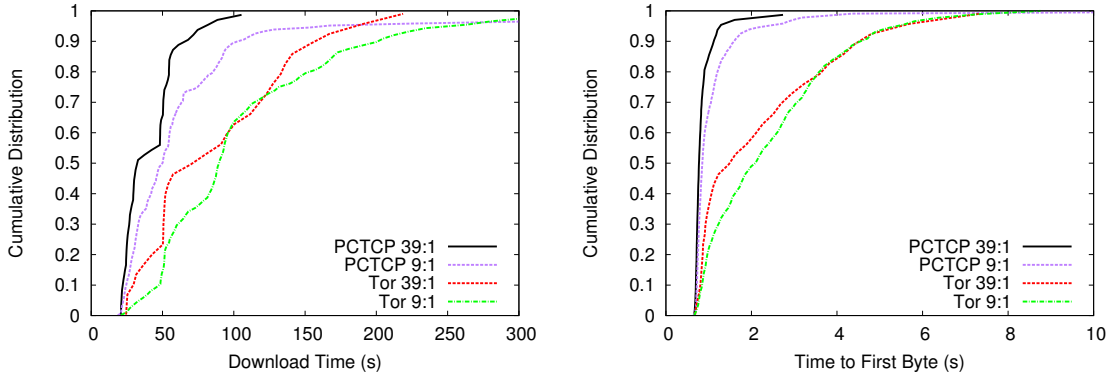
Figure 5.7: Performance of the bulk clients in a network of 500 clients and 50 routers with a 9:1 web-to-bulk client ratio. Compare to Figure 5.5.

significant improvements. The download time comparison between PCTCP and Tor for web and bulk clients, depicted in Figures 5.6(a) and 5.7(a), shows that PCTCP improves the long tail of the distribution for the web clients by approximately 20%. The reason for the improvement is that PCTCP allows each circuit at the transport layer to get its fair share of the bandwidth and forces the bulk downloads present in the system to back off whenever they attempt to get more than their allocated bandwidth, as evident by the degradation of the bulk client performance shown in Figure 5.7(a).



(a) (b)

Figure 5.8: Performance of the web clients in a high-bandwidth network of 400 clients and 20 routers. Compare to Figure 5.4.



(a) (b)

Figure 5.9: Performance of the bulk clients in a high-bandwidth network of 400 clients and 20 routers. Compare to Figure 5.5.

Table 5.1: Download time performance improvements at the median when PCTCP is used, as compared to Tor.

Client	*Light load (39:1)*	*High load (9:1)*
Web client	46%	65%
Bulk client	56%	44%

Table 5.2: Time-to-first-byte performance improvements at the median when PCTCP is used, as compared to Tor.

Client	*Light load (39:1)*	*High load (9:1)*
Web client	68%	73%
Bulk client	53%	61%

Figures 5.6(b) and 5.7(b) show the significant time-to-first-byte improvements for both the web clients and the bulk downloaders. The improvements at the 75th percentile are more than 60% for both the web and bulk clients.

5.5.2 Large-scale experiments using the topology presented in Section 3.4.1

We also experiment with PCTCP using the underlying Modelnet network we constructed and described in Section 3.4.1.[9] Our overlay Tor network is a scaled-down network in which we run 400 clients and 20 Tor routers. The ORs are assigned bandwidth capabilities that are sampled from the bandwidth distribution of the live Tor network ORs. We test the performance of PCTCP in this topology using a light traffic load of 39:1 web-to-bulk client ratio, and using a high traffic load of 9:1 web-to-bulk client ratio.

Figures 5.8 and 5.9 show the download time and time-to-first-byte comparisons for Tor and PCTCP using the different traffic loads for the web and bulk clients. The figures show that for Tor clients, the performance degrades faster, compared to PCTCP clients, as we increase the traffic load in the network by decreasing the web-to-bulk client ratio. For example, for the web client, the median time-to-first-byte remains 0.9 seconds for PCTCP under the low and high

[9]Note that, unlike our experiments in Section 3.4.1 where bulk clients model streaming users, our bulk clients here serve as traffic generators that continuously download 5 MiB files without pausing. The web clients download 320 KiB files and pause for a random time between 1 and 30 seconds between fetches.

traffic loads, whereas the corresponding value in Tor degrades by approximately 20%. This is also true for the download time distribution. The median download time for PCTCP remains the same as we increase the load (though the fourth quartile is slightly degraded), whereas the median download time for Tor clients degrades by more than 30%.

Because this network topology has more available bandwidth than the topology used in Section 5.5.1, more substantial performance benefits can be obtained when PCTCP is used, as summarized in Tables 5.1 and 5.2, because using separate TCP connections for each circuit allows each circuit to negotiate more bandwidth from the underlying network topology.

5.5.3 Live Experiments

To further test our new proposed design, we also conducted some experiments on the live network in October and November 2012. We next describe our experimental setup and then present our results.

Experimental Setup. Our setup is shown in Figures 5.10 and 5.11. Using OpenSwan [Ope13], we configured an IPsec connection between our two ORs, entry and middle, which we deployed on the live Tor network. Our entry implements PCTCP which can be enabled as a configuration option[10] only for our clients, so as not to affect other users of the network. Our middle OR runs an unmodified Tor process, but, as above, has an IPsec connection configured. For gathering Tor measurements, we simply turned off the option to enable PCTCP from the configuration of the entry. Both ORs have been configured with a bandwidth rate of 250 KB/s. To protect the privacy of other users, we configure both ORs to belong to the same *Tor family*, which prevents other users' unmodified Tor clients from choosing them both on one circuit. Also, we do not disable TLS in order to avoid risking other users' privacy in case of an accidental misconfiguration. We next describe our two experiments and present our results.

Experiment 1. In our first live experiment, we run four local web clients, which are configured to use our entry and middle ORs as their first two hops for all circuits constructed. The exit OR is chosen according to Tor's router selection algorithm from other ORs on the live network. Our clients download a fixed-sized 300 KB file from an external server and pause randomly for 3 to 30 seconds between downloads. We have also implemented the *MeasureMe* cell, as described in Section 4.5. For this particular experiment, our clients send this cell to the entry OR. This ensures that PCTCP is not used for other users' traffic, but only for our clients.

Results of Experiment 1. Our download time and time-to-first-byte results are shown in Figures 5.12(a) and 5.12(b). Both metrics show a substantial improvement for the clients using

[10] We used the latest stable version (0.2.2.39) for our live experiments.

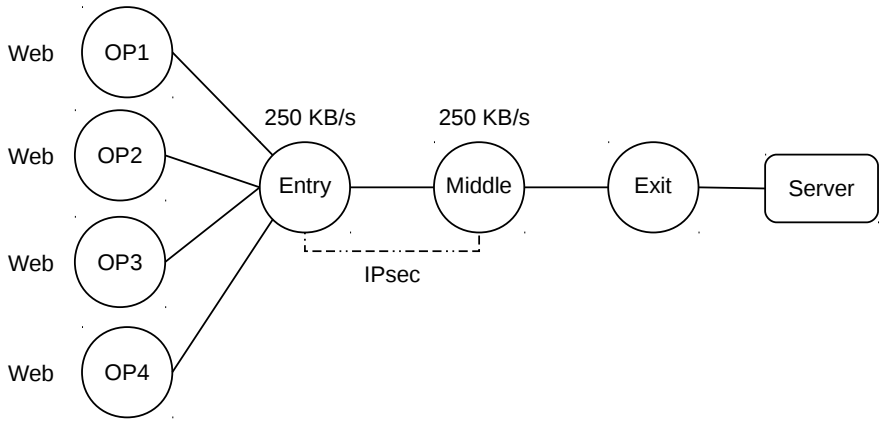


Figure 5.10: Setup for live experiment 1. The four web clients are configured to use the same entry and middle ORs for all of their circuits. The exit is chosen according to Tor's usual router selection algorithm. The entry and middle ORs communicate using an IPsec connection and are both configured with a bandwidth rate of 250 KB/s.

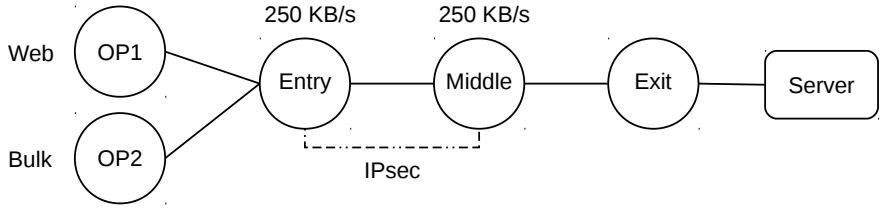
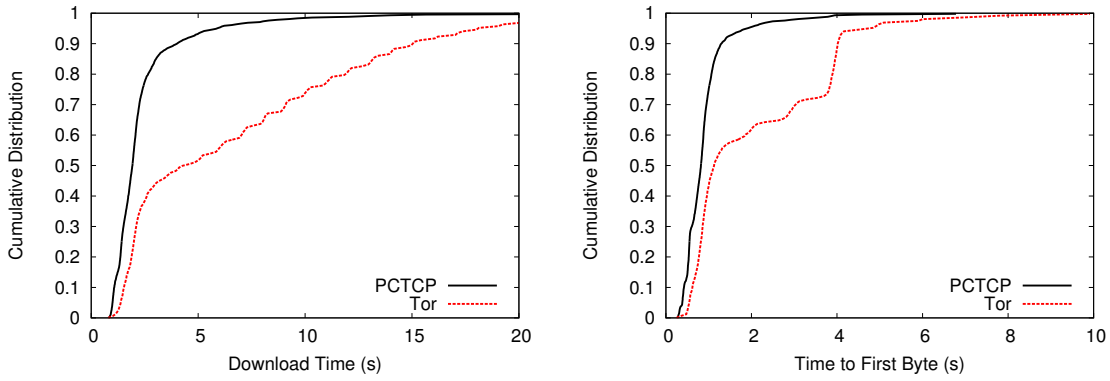


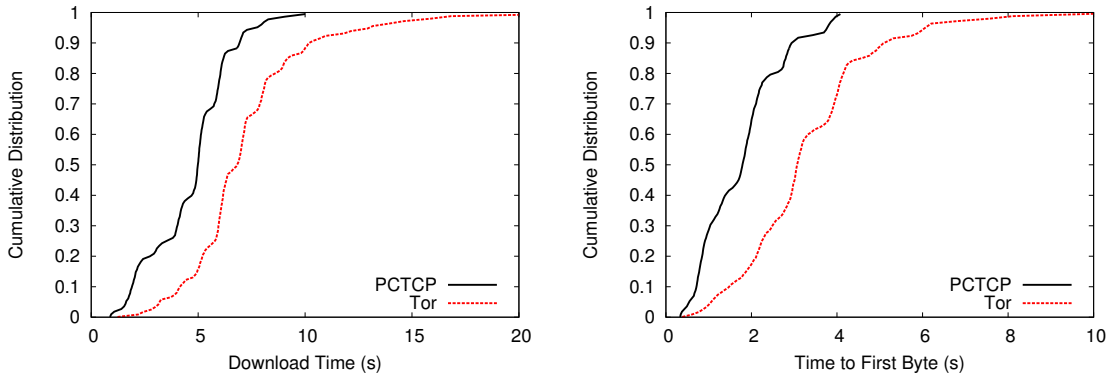
Figure 5.11: Setup for live experiment 2. The two (web and bulk) clients are configured to use the same entry and middle ORs for all their circuits. The exit is chosen according to Tor's usual router selection algorithm. The entry and middle ORs communicate using an IPsec connection and are both configured with a bandwidth rate of 250 KB/s.

PCTCP, as compared to Tor clients. Additionally, the performance distributions of PCTCP show a very slow degradation, and are much tighter, with smaller tails, compared to their Tor counterparts. Figure 5.12(a) compares the download time results for Tor and PCTCP. At the median, PCTCP clients finish downloading the file in 1.9 seconds, whereas Tor clients finish downloading the file in 4.3 seconds. This translates to an improvement of roughly 55%. The improvement jumps to 76% at the 75^{th} percentile.



(a) (b)

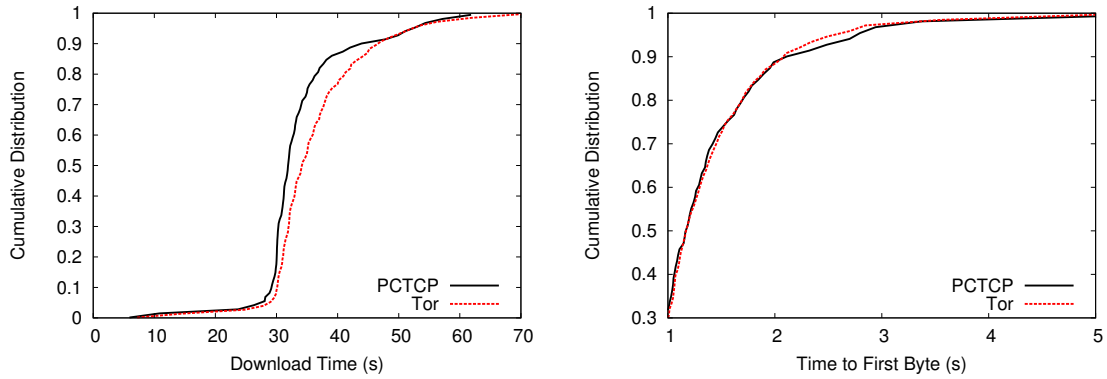
Figure 5.12: Results of live experiment 1, showing large performance benefits when PCTCP is used. Figure 5.12(a) depicts the download time performance for PCTCP and Tor. Figure 5.12(b) shows the time-to-first-byte performance for PCTCP and Tor.



(a) (b)

Figure 5.13: Results of live experiment 2 for the web client, showing improved performance when PCTCP is used. Figure 5.13(a) shows the download time comparison of PCTCP and Tor. Figure 5.13(b) shows the time-to-first-byte results for PCTCP and Tor.

As can be seen in Figure 5.12(b), at the median, there is a 27% improvement for the time-to-first byte when PCTCP is used. At the 75th percentile, the performance benefits are 74%; there, PCTCP successfully reduces the time-to-first-byte from 3.9 seconds to only 1 second.



(a) (b)

Figure 5.14: Results of live experiment 2 for the bulk client. Figure 5.14(a) shows the download time comparison of PCTCP and Tor. Figure 5.14(b) shows the time-to-first-byte results for PCTCP and Tor. Both figures show similar performance for PCTCP and Tor.

Experiment 2. The setup of our second experiment is similar to the first, except that we run two clients instead of four. One client acts as the bulk traffic generator by continuously downloading a 5 MB file without pausing between downloads. The second client is an interactive web browsing client that downloads a 300 KB file and pauses randomly for 3 to 30 seconds between downloads. Our clients also used the MeasureMe cell to ensure PCTCP is only used for their circuits.

Results of Experiment 2. Figure 5.13(a) depicts the download time performance for Tor and PCTCP for the web client.[11] With PCTCP, it takes 4.9 seconds to finish downloading, while Tor takes 6.8 seconds at the median. The improvements become more visible for the fourth quartile, as download times show a 26% improvement when PCTCP is used. Figure 5.13(b) shows the time-to-first-byte results for PCTCP and Tor. Again, the results consistently show strong improvements that are magnified at the third and fourth quartiles. For instance, at the 75th percentile, the time-to-first-byte for Tor clients is approximately 4 seconds, whereas for PCTCP clients, it is only 2.1 seconds, which is a more than 47% improvement.

Finally, Figure 5.14(a) demonstrates that the PCTCP bulk client exhibited slightly better performance than the Tor bulk client. Note that in this experiment, the introduction of the bulk downloader consumes the majority of the available bandwidth between entry and middle. Nevertheless, PCTCP still maintains the performance advantage for web clients compared to Tor. In

[11] Note that the stair-step pattern is a consequence of Tor's token bucket algorithm which flushes data once per second. This pattern becomes more visible with increased congestion. In versions of Tor more recent than the latest stable version we used, this flushing has been increased to ten times per second.

Figure 5.14(b), both PCTCP and Tor produced very similar fast time-to-first-byte results as the light web traffic did not introduce congestion to the bulk client.

From the previous experiments and results, one can make the following interesting observation: the amount of download time performance improvements achieved for a circuit using PCTCP highly depends on the available bandwidth between the two ORs that use PCTCP. That is, the more the available bandwidth between two ORs that use PCTCP between them exists, the more the download time enhancements will be observed with PCTCP.

For example, in experiment 1, we observed download time improvements that start at 76% for the fourth quartile, whereas for experiment 2, the respective improvements start at 26%. The main difference between the two experiments is that the bulk client consumed the majority of the bandwidth in experiment 2, leaving less room for improvements for the web client. However, we observe that PCTCP produces significantly smaller time-to-first-byte delays than Tor, regardless of the congestion state between the ORs.

Based on these observations, we conclude that PCTCP produces performance benefits that can certainly be perceived by clients. To maximize the benefits of using PCTCP, we believe it should be used in combination with previous proposals that aim to increase the amount of available bandwidth in the network, such as traffic classification (Chapter 4), throttling approaches [MWS11, JSH12] or approaches aimed to incentivize clients to run ORs [JHK10, NDW10].

5.6 Discussion

We next discuss a variety of open issues regarding PCTCP.

5.6.1 Anonymity Implications

Since our transport proposal is designed for Tor, an anonymity network, it is essential to consider the anonymity implications of our design. In particular, it is important to ensure that our new design does not add new vulnerabilities to the Tor network. Recall that the anonymity of a circuit is compromised in Tor if its two ends, the entry and exit, are compromised. Therefore, one issue to consider is whether using PCTCP can reduce the anonymity set of the ORs used in a circuit. For example, can an exit OR reduce the anonymity set of the entry OR used on a circuit because of PCTCP?

First, with exception of the IPsec connections, the changes that are imposed by PCTCP on any OR are local. That is, our design does not introduce a new cell type or require other ORs on the circuit to upgrade. If an entry OR uses PCTCP, then only the middle OR will notice because the middle has to agree to establish the IPsec connection with entry and because it receives more than one TCP connection from the upgraded entry. Those changes do not affect the exit OR in the circuit; therefore, the exit would not be able to know if entry belongs to the set of upgraded ORs or not. Even if the exit learns from router descriptors that middle is an upgraded OR, the exit would still not be able to know if entry is upgraded or not. Therefore, we believe that PCTCP does not introduce any new threats to the Tor network.

Furthermore, one might wonder if dedicating separate TCP connections might open the door to timing attacks. First, a connection between the OP and the OR is very similar for Tor and PCTCP. Second, because the communication between ORs is protected using IPSec, it would be difficult for the adversary to extract specific circuit information even though each circuit uses a separate TCP connection. Therefore, we believe that PCTCP does not introduce any new threats to the Tor network.

5.6.2 Incremental Deployment

One advantage of PCTCP is that it is incrementally deployable in two steps. The first step towards deployment is enabling IPsec communication among ORs. Basically, ORs need to advertise in their descriptors that they are willing to accept IPsec connections. Then, IPsec-enabled ORs can try to establish IPsec connections proactively among each other. When OR1 wishes to use PCTCP with OR2, it can check if it has an existing IPsec connection with OR2,[12] in which case OR1 can proceed with using PCTCP. If OR1 detects no IPsec connection with OR2, it uses the default Tor TLS connection with OR2 and multiplexes the circuits in the same connection.

5.6.3 Experimental Limitations

To be able to faithfully test and evaluate our new transport proposal, we ran a series of testbed experiments on different network topologies using different traffic models and loads. Regardless of our efforts, we recognize that our large-scale experiments were conducted on an isolated experimental testbed. We were unable to experiment with larger topologies because we are limited by our CPU, bandwidth and memory resources.

[12] For Openswan, the visibility of IPsec for an application can be established using libwhack.

However, to ensure that we report accurate results, we followed the methodology of Jansen *et al.* [JBHD12] to produce an accurate model of the Tor network. We also used their published topology files in order to avoid biased results that might be obtained using a different experimental setup. Finally, we carried out additional experiments on the live Tor network to confirm our results.

Another experimental difficulty we faced is running IPsec on ExperimenTor. Our large-scale experiments that tested PCTCP did not use IPsec; however, we have not disabled the TLS encryptions in our PCTCP experiments to maintain a by-hop layer of encryption. While we do not expect TLS and IPsec to have the same exact performance, we believe that the slight difference in performance between TLS and IPsec would not impact the validity of our large-scale experimental results. This is evident in the results revealed by our live network experiments, in which we used an IPsec connection between the first two ORs.

5.6.4 IPsec through NATs

One challenge that IPsec faced in the past is its inability to connect to hosts behind NATs. As a result, NAT-Traversal [KHSV05] (NAT-T) has evolved to address this problem. NAT-T can be used when two hosts detect that if they are behind a NAT. In the context of Tor, we believe this problem is currently irrelevant as most Tor ORs are publicly reachable; however, there are some efforts to enable the operation of ORs from behind NATs [App12]. In this case, IPsec can still benefit from NAT-T.

5.6.5 File Descriptor and Memory Usage

One issue to consider is how this work affects the very busy routers on the live network. Since Tor uses a weighted-bandwidth OR selection algorithm where ORs are selected in proportion to their bandwidth, some high-bandwidth ORs service thousands of circuits at the same time. This means that, with PCTCP, such routers are expected to maintain thousands of file descriptors at the same time. One might wonder if such a requirement might raise memory usage concerns due to the TCP buffer space allocated in the kernel for each file descriptor.

To get an idea of how many file descriptors would be needed when PCTCP is used, we examined a fast exit OR on the live network configured with a bandwidth of 100 Mb/s, which puts it among the fastest 6% of the network routers. This fast exit OR used roughly 10,000 file descriptors for its communication with other ORs and with destination servers. Since an exit OR uses one file descriptor for each *stream* within a circuit, the number of circuits it is handling is

certainly less than the number of file descriptors it is using. Note that intermediate routers are currently expected to use a number of file descriptors that is equal to the number of ORs in the network, which is approximately 3000. We therefore expect that other intermediate ORs, such as middles or entries that have the same bandwidth capabilities as the fast exit, to use between 3000 and 10,000 file descriptors if they use PCTCP. In short, file descriptor and memory usage should not be a problem with PCTCP, as even the busiest entry and middle ORs running PCTCP should consume fewer of these resources than the existing Tor network requires exit ORs to support today.

5.6.6 Future Work

One important area for future investigation is to implement other transport proposals such as TCP-over-DTLS and UDP-OR, in order to compare their performance to that of PCTCP in large-scale network emulation. Tor's forthcoming transport abstraction layer [She12] should greatly facilitate this task.

Another area for future work is to consider an alternative queueing design for Tor that reduces the number of times cells are copied. Indeed, our design eliminates the need for circuit queues as every input buffer corresponds to single output buffer, which means that data can be copied immediately from the input buffer to the output buffer after being encrypted or decrypted.

5.7 Conclusion

In this work, we recognize the importance of the Tor network as a privacy-preserving tool online and seek to enhance its performance for interactive application users. To this end, we propose PCTCP, a new anonymous communication transport design for Tor which allows every circuit to use a separate kernel-level TCP connection which is protected by IPsec. Our design is easily deployable and requires minimal changes to routers. Furthermore, experimental evaluation of PCTCP shows vast improvement gains, while maintaining the threat model of the Tor network. Our live experiments show that it is possible to obtain improvements of more than 74% for response times and more than 76% for download times when PCTCP is used compared to Tor.

Chapter 6

DefenestraTor: Throwing out Windows in Tor

An earlier version of this chapter appeared in the 11th Privacy Enhancing Technologies Symposium [ABG+11].

In this chapter, we look at the congestion control mechanism in Tor and seek to enhance its performance by offering techniques to control congestion and improve flow control in order to reduce unnecessary delays. We first evaluate small fixed-size circuit windows and a dynamic circuit window that adaptively resizes in response to perceived congestion. While these solutions improve web page response times and require modification only to exit routers, they generally offer poor flow control and slower downloads relative to Tor's current design. To improve flow control while reducing congestion, we implement *N23*, an ATM-style per-link algorithm that allows Tor routers to explicitly cap their queue lengths and signal congestion via back-pressure. Our results show that N23 offers better congestion and flow control, resulting in improved web page response times and faster page loads compared to Tor's current design and the other window-based approaches. We also argue that our proposals do not enable any new attacks on Tor users' privacy.

6.1 Introduction

As discussed in chapter 1, one of the most significant road blocks to Tor adoption is its excessively high and variable delays, which inhibit interactive applications such as web browsing.

Many prior studies have diagnosed a variety of causes of this high latency (see Dingledine and Murdoch [DM09] for a concise summary). Most of these studies have noted that the queuing delays often dominate the network latencies of routing packets through the three routers. These high queuing delays are, in part, caused by bandwidth bottlenecks that exist along a client's chosen circuit. As high-bandwidth routers forward traffic to lower-bandwidth downstream routers, the high-bandwidth router may be able to read data faster than it can write it. Because Tor currently has no explicit signaling mechanism to notify senders of this congestion, packets must be queued along the circuit, introducing potentially long and unnecessary delays for clients. While recent proposals seek to re-engineer Tor's transport design, in part, to improve its ability to handle congestion [Vie08, KBC08, RG09], these proposals face significant deployment challenges, as discussed in Chapter 5.

Improving congestion and flow Control. To reduce the delays introduced by uncontrolled congestion in Tor, we design, implement, and evaluate two classes of congestion and flow control. First, we leverage Tor's existing end-to-end window-based flow control framework and evaluate the performance benefits of using small fixed-size circuit windows, reducing the amount of data in flight that may contribute to congestion. We also design and implement a dynamic window resizing algorithm that uses increases in end-to-end circuit round-trip time as an implicit signal of incipient congestion. Similar solutions are being considered for adoption in Tor to help relieve congestion [Din09], and we offer a critical analysis to help inform the discussion. Window-based solutions are appealing, since they require modifications only to exit routers.

Second, we offer a fresh approach to congestion and flow control inspired by standard techniques from Asynchronous Transfer Mode (ATM) networks. We implement a per-link credit-based flow control algorithm called N23 [KBC94] that allows Tor routers to explicitly bound their queues and signal congestion via back-pressure, reducing unnecessary delays and memory consumption. While N23 offers these benefits over the window-based approaches, its road to deployment may be slower, as it may require all routers along a circuit to upgrade.

Evaluation. We conduct a holistic experimental performance evaluation of the proposed algorithms with realistic traffic models. We show that the window-based approaches offer up to 65% faster web page response times relative to Tor's current design. However, they offer poor flow control, causing bandwidth under-utilization and ultimately resulting in poor download time. In contrast, our N23 experiments show that delay-sensitive web clients experience up to 65% faster web page responses and a 32% decrease in web page load times compared to Tor's current design.

6.2 Tor's Approach to Congestion and Flow Control

Since the Tor network consists of volunteer-run routers from across the world, these routers have varying and often limited amounts of bandwidth available to relay Tor traffic. Consequently, as clients choose their circuits, some routers have large amounts of bandwidth to offer, while others may be bandwidth bottlenecks. In order for Tor to offer the highest degree of performance possible, it is necessary to have effective mechanisms in place to ensure steady flow control, while also detecting and controlling congestion. Recall that flow control regulates the transmission rate between a sender and a receiver so that the receiver is not overwhelmed with data, whereas congestion control is concerned with controlling congestion at the intermediate routers in the core network. In this section, we discuss the many features that directly or indirectly impact congestion and flow control in Tor.

6.2.1 Congestion and Flow Control Mechanisms

Pairwise TCP. All packets sent between Tor routers are guaranteed to be delivered reliably and in-order by using TCP transport. As a result of using TCP, communications between routers can be protected with TLS link encryption.

Circuit and stream windows. Recall that Tor uses two layers of end-to-end window-based flow control between the exit router and the client to ensure steady flow control (discussed in Section 2.3.3).

Both the stream-level and circuit-level windows are relatively large and static. To illustrate how this can degrade performance, consider the following scenario. Suppose a client downloads files through a circuit consisting of 10 MiB/s entry and exit routers and a 128 KiB/s middle router. Since the exit router can read data from the destination server faster than it can write it to its outgoing connection with the middle router, and the reliable TCP semantics preclude routers from dropping cells to signal congestion, the exit router must buffer up to one full circuit window (500 KiB) worth of cells. Furthermore, as shown in Figure 6.1, these cells often sit idly for several seconds while the buffer is slowly emptied as `SENDME` cells are received. Since cells may travel down a circuit in large groups of up to 500 KiB followed by periods of silence while the exit router waits for `SENDME` replies, Tor's window-based flow control does not always keep a steady flow of cells in flight.

Token bucket rate limiting. In order to allow routers to set limits on the amount of bandwidth they wish to devote to transiting Tor traffic, Tor offers token bucket rate limiting. Briefly, a router starts with a fixed amount of tokens, and decrements their token count as cells are sent or received. When the router's token count reaches zero, the router must wait to send or receive

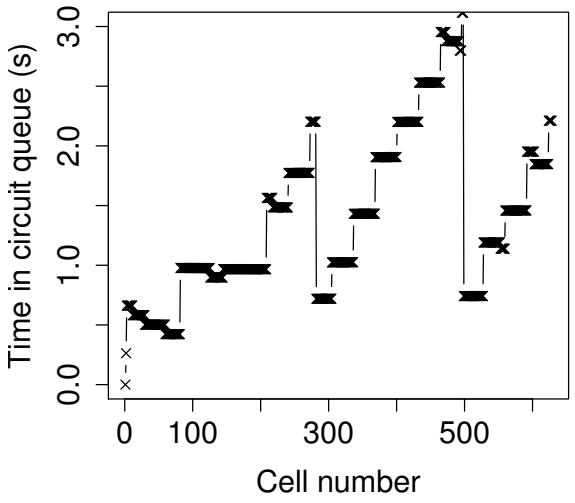


Figure 6.1: The exit router's circuit queue delays for a 300 KiB download

until the tokens are refilled. To reduce Tor's CPU utilization, tokens used to be refilled only once per second, but in more recent releases of Tor, tokens are refilled ten times per second. It has been previously observed that refilling the tokens so infrequently contributes in part to Tor's overall delays [DSR+10].

Circuit-building timeouts. Tor has incorporated adaptive circuit-building timeouts that measure the time it takes to build a circuit, and eliminate circuits that take an excessively long time to construct [CP08]. The intuition is that circuits that build slowly are highly congested, and would in turn offer the user poor performance. While this approach likely improves the users' quality of service in some cases, it does not help to relieve congestion that may occur at one or more of the routers on a circuit *after* the circuit has been constructed.

6.3 Improving Tor's Congestion and Flow Control

Our primary goal is to improve Tor's performance, specifically by better understanding and improving Tor's congestion and flow control. We consider two broad classes of solutions. First, we wish to understand how much improvement is possible simply by adjusting Tor's existing end-to-end window-based flow control mechanisms to reduce the amount of data in flight, and

thereby mitigate congestion. We also evaluate an end-to-end congestion control technique that enables exit Tor routers to infer incipient congestion by regarding increases in end-to-end round-trip time as a congestion signal. Second, we consider a fresh approach to congestion and flow control in Tor, eliminating Tor's end-to-end window-based flow control entirely, and replacing it with ATM-style, per-link flow control that caps routers' queue lengths and applies back-pressure to upstream routers to signal congestion.

6.3.1 Improving Tor's Existing End-to-End Flow Control

We first consider whether adjusting Tor's current window-based flow control can offer significant performance improvements. Keeping Tor's window-based mechanisms is appealing, as solutions based on Tor's existing flow control framework may be deployed immediately, requiring modifications only to the exit routers, not clients or non-exit routers.

Small Fixed-size Circuit Windows. The smallest circuit window size possible without requiring both senders and receivers to upgrade is 50 KiB (100 cells, or one circuit-level `SENDME` interval). We evaluate how fixed 50 KiB circuit windows impact clients' performance.

Dynamic Circuit Windows. We next consider an algorithm that initially starts with a small, fixed circuit window and dynamically increases the window size (*e.g.*, amount of unacknowledged data allowed to be in flight) in response to positive end-to-end latency feedback. Inspired by latency-informed congestion control techniques for IP networks [BOP94, WC92], we propose an algorithm that uses increases in perceived end-to-end circuit round-trip time (RTT) as a signal of incipient congestion.

The algorithm works as follows. Initially, each circuit's window size starts at 100 cells. First, the sender calculates the circuit's end-to-end RTT using the circuit-level `SENDME` cells, maintaining the minimum RTT (rtt_{min}) and maximum RTT (rtt_{max}) observed for each circuit. We note that rtt_{min} is an approximation of the base RTT, where there is little or no congestion on the circuit. Next, since RTT feedback is available for every 100 cells, the circuit window size is adjusted quickly using an additive increase, multiplicative decrease (AIMD) window scaling mechanism based on whether the current RTT measurement (rtt) is less than the threshold T, defined in Equation 6.1. This threshold defines the circuit's tolerance to perceived congestion.

$$T = (1 - \alpha) \times rtt_{min} + \alpha \times rtt_{max} \tag{6.1}$$

Choosing a small α value ensures that the threshold is close to the base RTT, and any increases beyond the threshold implies the presence of congestion along the circuit.[4] For each RTT mea-

[4]For our experiments, we use $\alpha = 0.25$.

surement (*e.g.*, each received circuit-level SENDME), the circuit window size (in cells) is adjusted according to Equation 6.2.

$$new\_window(rtt) = \begin{cases} old\_window + 100 & \text{if } rtt \leq T \\ \lfloor old\_window/2 \rfloor & \text{otherwise} \end{cases} \tag{6.2}$$

Finally, we explicitly cap the minimum and maximum circuit window sizes at 100 and 1000 cells, respectively.[5]

6.3.2 ATM-style Congestion and Flow Control for Tor

Because Tor's flow control works at the circuit's edges—the client and the exit router—we seek to improve performance by implementing per-link flow control to ensure a steady flow of cells while reducing congestion at the intermediate routers. Implementing per-link flow control in Tor resembles the problem of link-by-link flow control (LLFC) in ATM networks. While the goals of Tor and ATM are certainly different, there are many similarities. Both networks are connection-oriented, in the sense that before applications can send or receive data, virtual circuits are constructed across multiple routers or switches, and both have fixed-sized cells. Furthermore, it has been shown that ATM's credit-based flow control approaches, such as the N23 scheme, eliminate cell loss due to buffer overflows [Jai95], a feature that makes such approaches similar to Tor, where no packets may be dropped to signal congestion.

N23 Flow Control for Tor. Figure 6.2 depicts the N23 scheme that we integrated into Tor, and it works as follows. First, when a circuit is built, each router along the circuit is assigned an initial *credit balance* of $N2 + N3$ cells, where $N2$ and $N3$ are system parameters. $N2$ cells is the available steady state buffering per circuit, $N3$ cells is the allowed excess buffering, and the circuit's queue length is strictly upper bounded by $N2 + N3$ cells. In general, $N2$ is fixed at the system's configuration time, but $N3$ may change over a circuit's lifetime.

When a router forwards a cell, it decrements its credit balance by one for that cell's circuit. Each router stops forwarding cells if its credit balance reaches zero. Thus, routers' circuit queues are upper bounded by $N2+N3$ cells, and congestion is indicated to upstream routers through this back-pressure. Next, for every $N2$ cells forwarded, the downstream router sends a *flow control cell* to the upstream router that contains credit information reflecting its available circuit queue

[5]Note that a selfish Tor client could attempt to increase their circuit window by pre-emptively acknowledging data segments before they are actually received. Prior work in mitigating similar behavior in selfish TCP receivers may be applied here [SCWA99, SBB05].

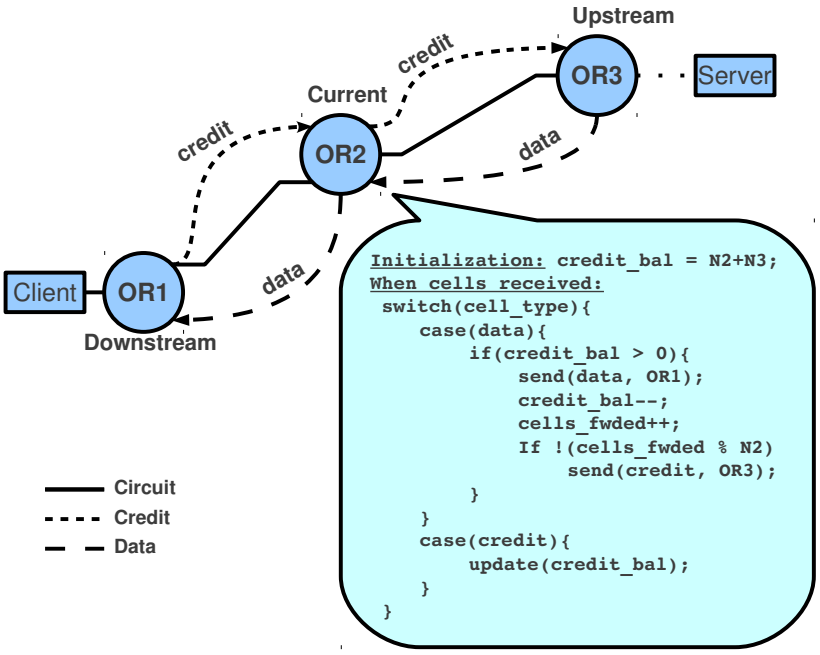


Figure 6.2: N23 credit-based flow control in Tor

space. On receiving a flow control cell, the upstream router updates the circuit's credit balance and may forward cells only if the credit balance is greater than zero.

Adaptive Buffer Sizes. The algorithm as described assumes a static $N3$. We also developed an adaptive algorithm that reduces the $N3$ value when there is downstream congestion, which is detected by monitoring the delay that cells experience in the connection's output buffer. When the congestion subsides, $N3$ can increase again. The value of $N3$ is updated periodically and is bounded by a minimum and a maximum value (100 and 500 cells, respectively).

Advantages. The N23 algorithm has two important advantages over Tor's current flow control. First, the size of the circuit queue is explicitly capped, and guaranteed to be no more than $N2 + N3$ cells. This also ensures steady flow control, as routers typically have cells available to forward. Tor's current flow control algorithm allows the circuit queue of a circuit's intermediate routers to grow up to one circuit window in size, which not only wastes memory, but also results

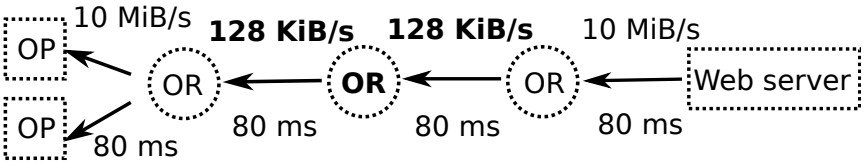


Figure 6.3: A simple topology with a middle router bandwidth bottleneck

in unnecessary delays due to congestion. In contrast, for typical parameter values ($N3 = 70$ and $N2 = 20$), N23 ensures a strict circuit queue bound of 90 cells, while these queues currently can grow up to 1000 cells in length.

The second advantage is that adaptive N3 reacts to congestion within a single link RTT. When congestion occurs at a router, the preceding router in the circuit will run out of credit and must stop forwarding until it gets a flow control cell.

6.4 Experiments and Results

To demonstrate the efficacy of our proposed improvements, we offer a whole-network evaluation of our congestion and flow control algorithms using ExperimenTor, which is based on Modelnet. Recall that Modelnet enables the experimenter to specify realistic network topologies annotated with bandwidth, delay and other link properties, and run real code on the emulated network.

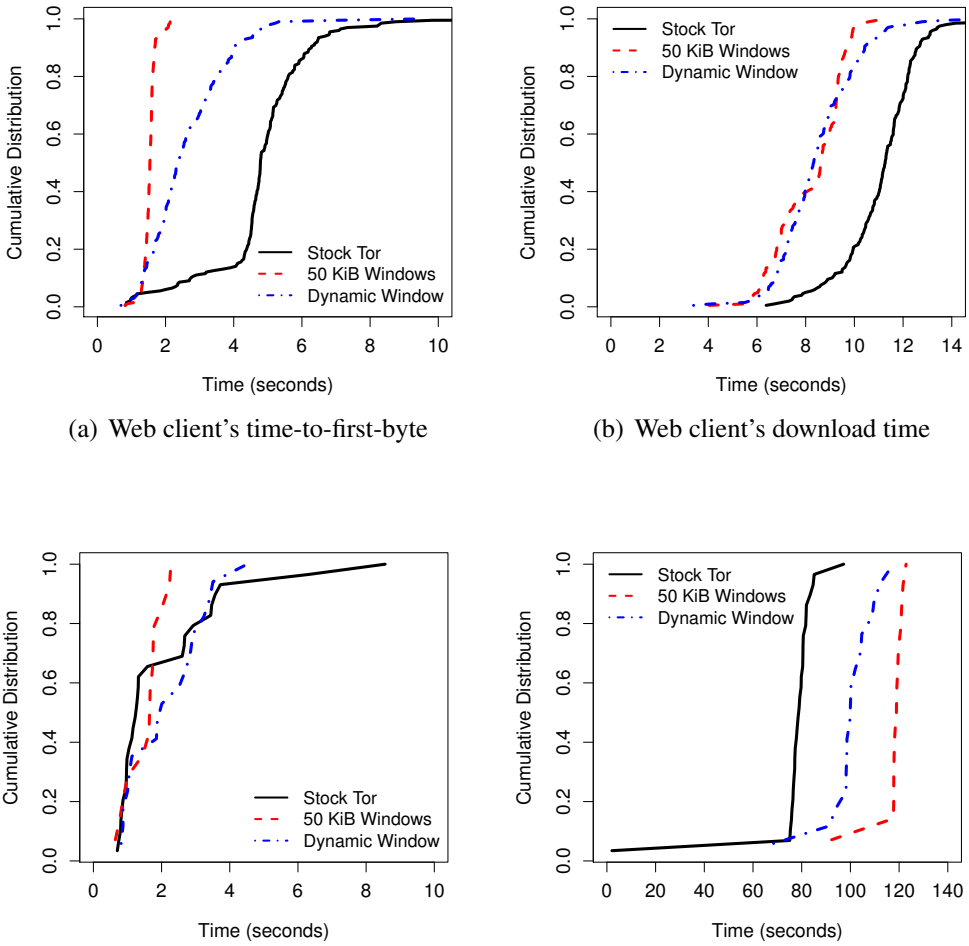
Our evaluation focuses on performance metrics that are particularly important to the end-user's quality of service. First, we measure the time-to-first-byte and the overall download time. For our experiments in the chapter, we use a development branch of the Tor source code (version `0.2.3.0-alpha-dev`).[7]

6.4.1 Small-scale Analysis

Setup. We emulate the topology depicted in Figure 6.3 on ModelNet where two Tor clients compete for service on the same set of routers with a bandwidth bottleneck at the middle router.[8]

[7] In our evaluation, we refer to unmodified Tor version `0.2.3.0-alpha-dev` as *stock Tor*, 50 KiB (100 cell) fixed windows as *50 KiB window*, the dynamic window scaling algorithm as *dynamic window*, and the N23 algorithm as *N23*.

[8] Note that a 128 KiB/s router corresponds to the 65th percentile of routers ranked by observed bandwidth, as reported by the directory authorities. Thus, it is likely to be chosen fairly often by clients. Also, 80 ms is roughly the median measured latency between end-hosts according to the King data set [GKL+09].



(a) Web client's time-to-first-byte

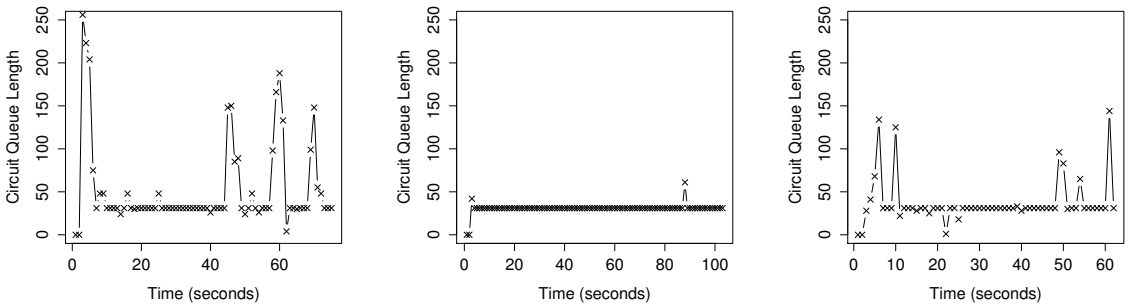
(b) Web client's download time

(c) Bulk client's time-to-first-byte

(d) Bulk client's download time

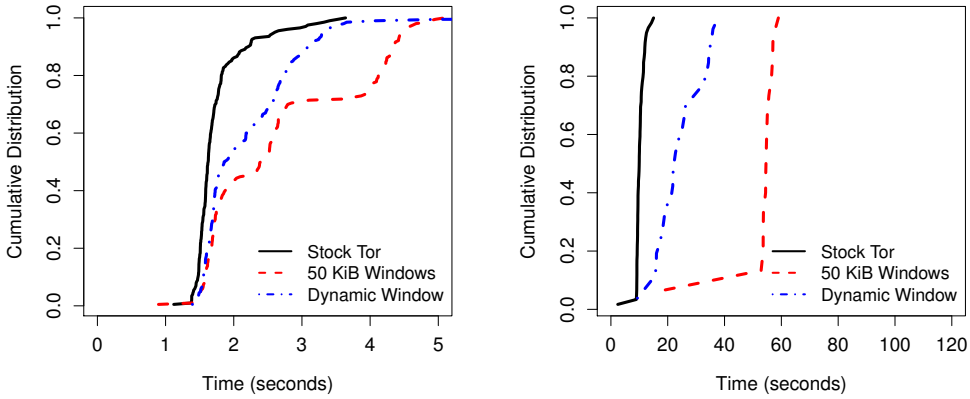
Figure 6.4: Performance comparisons for window approaches in a bottleneck topology

One client downloads 300 KiB, which corresponds to the size of an average web page [Ram12]. The second client, a bulk downloader, fetches 5 MiB. Both clients pause for a random amount of time between one and three seconds, and repeat their downloads. Each experiment concludes after the web client completes 200 downloads. Both clients use the `wget` web browser and the destination runs the `lighthttpd` web server.



(a) Stock Tor (b) 50 KiB window (c) Dynamic window

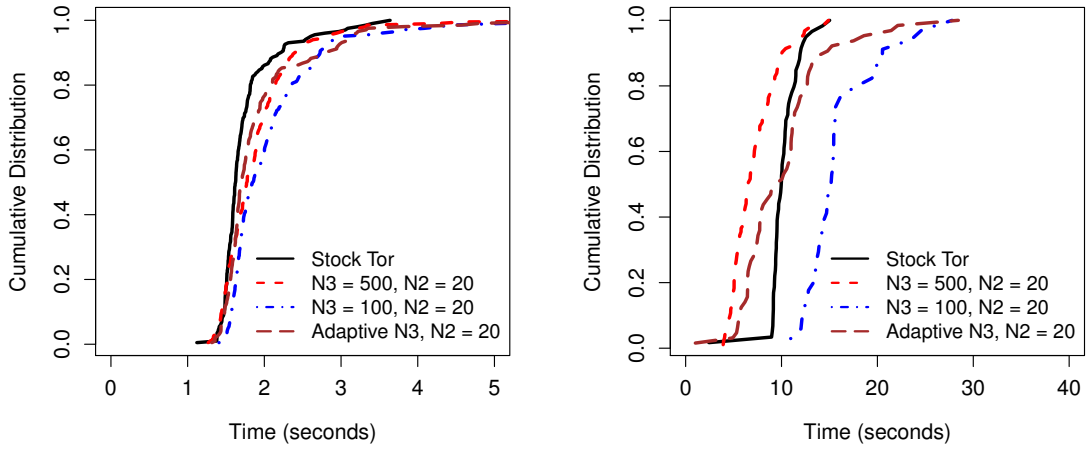
Figure 6.5: Bulk client's circuit queues at the exit router over the course of a download



(a) Web client's download time (b) Bulk client's download time

Figure 6.6: Performance comparisons for window approaches in a non-bottleneck topology

End-to-end Window-based Solutions. Figure 6.4(a) shows that the time-to-first-byte for a typical web client using stock Tor is 4.5 seconds at the median, which is unacceptably high for delay-sensitive, interactive web users who must incur this delay for each web request. In addition, stock Tor's circuit queues fluctuate in length, growing up to 250 cells long, and remaining long for many seconds, indicating queuing delays, as shown in Figure 6.5(a). Reducing the circuit window size to 50 KiB (*e.g.*, one circuit `SENDME` interval) offers a median time-to-first-byte of less than 1.5 seconds, and dynamic windows offer a median time-to-first-byte of two seconds.



(a) Web client's download time

(b) Bulk client's download time

Figure 6.7: Download time comparison for Tor and N23 in a non-bottleneck network

In Figure 6.4(b), we see that the web client's download time is influenced by the high time-to-first-byte, and is roughly 40% faster with 50 KiB and dynamic windows relative to stock Tor. Also, the circuit queues are smaller with the 50 KiB and dynamic windows (see Figures 6.5(b) and 6.5(c)).

The bulk client experiences significantly smaller time-to-first-byte delays (in Figure 6.4(c)) than the web client using stock Tor. This highlights an inherent unfairness during congestion: web clients' traffic is queued behind the bulk traffic and, consequently, delay-sensitive clients must wait longer than delay-insensitive bulk downloaders to receive their first byte of data. Using a small or dynamic window reduces this unfairness, since the bound on the number of unacknowledged cells allowed to be in flight is lower.

However, Figure 6.4(d) indicates that the bulk client's download takes significantly longer to complete with 50 KiB windows relative to stock Tor. Thus, 50 KiB windows enhance performance for web clients at the cost of slower downloads for bulk clients. The bulk clients experience slower downloads because they keep less data in flight and, consequently, must incur additional round-trip time delays to complete the download. Dynamic windows offer a middle-ground solution, as they ameliorate this limitation by offering an improvement in download time for web clients while penalizing bulk clients less than small windows, but bulk clients are still penalized relative to stock Tor's performance.

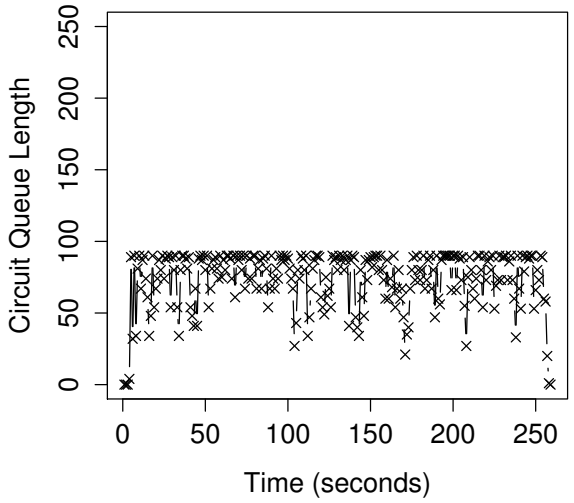


Figure 6.8: Circuit queue length with bottleneck: N3 = 70, N2 = 20

We next consider the same topology shown in Figure 6.3, except we replace the bottleneck middle router with a fast 10 MiB/s router. Because the 50 KiB and dynamic windows generally keep less data in flight, these solutions offer slower downloads relative to stock Tor, as shown in Figures 6.6(a) and 6.6(b).

Despite the improvements in time-to-first-byte in the presence of bandwidth bottlenecks, we find that smaller circuit windows tend to under-utilize the available bandwidth and the dynamic window scaling algorithm is unable to adjust the window size fast enough, as it receives congestion feedback infrequently (only every 100 cells). Also, even in the non-bottleneck topology, the 50 KiB window web client's time-to-first-byte is higher than the optimal delay from two circuit RTTs, which is 0.64 s. Lastly, 50 KiB windows offer worse flow control than Tor's current design, since only 50 KiB can be in flight, and the exit router must wait for a full circuit RTT until more data can be read and sent down the circuit.

Based on these drawbacks, we conclude that in order to achieve an improvement in both time-to-first-byte and download speed, it is necessary to re-design Tor's fundamental congestion and flow control mechanisms. We next offer an evaluation of per-link congestion and flow control for Tor.

Per-link Congestion and Flow Control. We first implemented N23 with fixed values of N2 and N3 (*static N23*) and then with N3 values that react to network feedback (*adaptive N3*). We

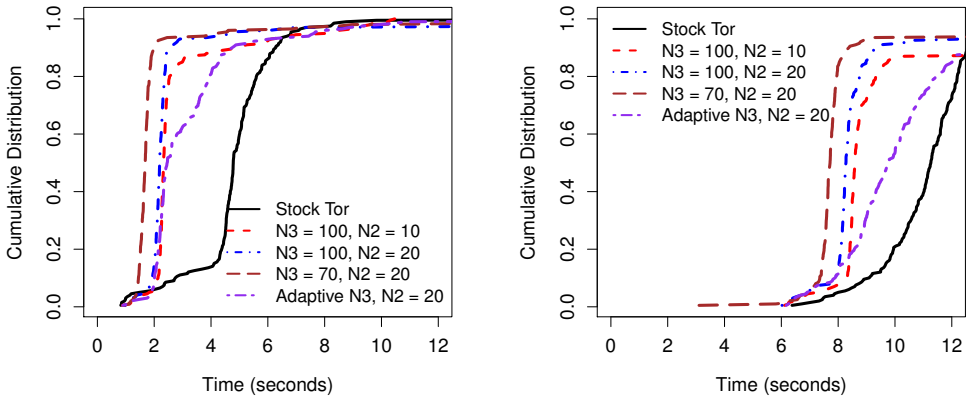
disabled Tor's window-based flow control, so that exit routers ignored SENDMEs they received from clients. We discuss the results of adaptive N3 with our large-scale experiments. In this section, we present the results of N23 for both the bottleneck and non-bottleneck topologies.

For the non-bottleneck topology, we see in Figure 6.7(b) that N23 provides a substantial improvement in download time for the 5 MiB downloads compared to stock Tor only for higher values of N3 — 500 cells, comparable to stock Tor's stream window size. The graph shows that there is a 25% decrease in delay for 50% of the bulk downloads when N23 is used. Since the maximum throughput is bounded by W/RTT, where W is the link's TCP window size and RTT is the link's round-trip time, and since N23's per-link RTT is significantly smaller than a stock Tor's complete circuit RTT, throughput is increased when N23 is used. This improvement suggests that in non-bottleneck scenarios, bulk traffic data cells are unnecessarily slowed down by Tor's flow control at the edges of the circuit. For bursty web traffic, both Tor's current flow control and N23 have similar performance for fixed and adaptive N3, as shown in Figure 6.7(a).

For bottleneck scenarios, Figures 6.9(a) and 6.9(b) show that smaller values of N3 improve both the download time and time-to-first-byte for the bursty web traffic. For example, the web browsing client experiences a 20% decrease in download time for 80% of the requests when N23 is used. Also, the web client's time-to-first-byte is only two seconds for 90% of the requests, whereas for the stock Tor client, 80% of web requests take more than four seconds to receive the first byte. Figure 6.8 shows that the circuit queue length is upper bounded by $N2 + N3 = 90$ cells.

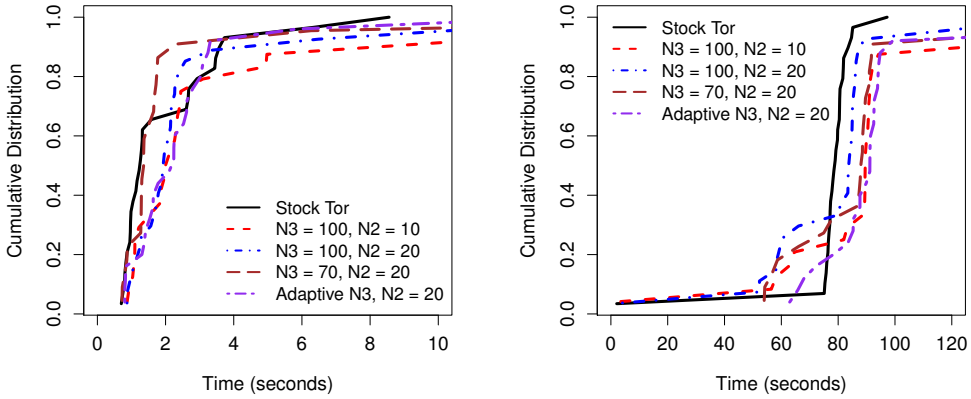
To understand how N23 performs with different $N2$ values, we repeated the bottleneck experiments while varying that parameter. Although a higher value for $N2$ has the undesirable effect of enlarging the circuit buffer, it can be seen in Figures 6.9(a) and 6.9(b) that when $N3$ is fixed at 100 cells, increasing $N2$ to 20 cells slightly improves both download time and time-to-first-byte. It can be observed from Figure 6.9(a) that time-to-first-byte is significantly improved by keeping a smaller $N3 = 70$ and a larger $N2 = 20$. Decreasing $N3$ to 70 cells makes up for the increase in the $N2$ zone of the buffer, which means we gain the benefits of less flow control overhead, and the benefits of a small buffer of $N2 + N3 = 90$ cells. While performance is improved for the web client, the bulk client's time-to-first-byte is not affected greatly, as seen in Figure 6.9(c), but its downloads generally take longer to complete, as we see in Figure 6.9(d). In addition, adaptive N3 offers improved time-to-first-byte and download times for the web client, while slowing downloads for the bulk client. By N23 restricting the amount of data in flight, the bandwidth consumed by bulk clients is reduced, improving time-to-first-byte and download time for delay-sensitive web clients.

Finally, the bandwidth cost associated with the N23 scheme is relatively low. For instance, with $N2 = 10$, a flow control cell must be sent by each router on the circuit for every 10 data



(a) Web client's time-to-first-byte

(b) Web client's download time



(c) Bulk client's time-to-first-byte

(d) Bulk client's download time

Figure 6.9: Performance comparisons for Tor and N23 in a bottleneck topology

cells forwarded, which requires a 10% bandwidth overhead per router. For $N2 = 20$, a flow control cell is sent for every 20 data cells, which is only a 5% overhead per router. While this cost is higher than Tor's window-based flow control (*e.g.*, one stream-level `SENDME` for every 50 data cells is only a 2% overhead per circuit), the cost of N23 is nonetheless modest.

6.4.2 Larger-scale Experiments

Setup. We next evaluate the window-based solutions and N23 with adaptive N3 in a more realistic network topology. We deploy 20 Tor routers on a random ModelNet topology whose bandwidths are assigned by sampling from the live Tor network. Each link's latency is set to 80 ms. Next, to generate a traffic workload, we run 200 Tor clients. Of these, ten clients are bulk downloaders who fetch files between 1–5 MiB, pausing for up to two seconds between fetches. The remaining 190 clients are web clients, who download files between 100–500 KiB (typical web page sizes), pausing for up to 30 seconds between fetches. This proportion of bulk-to-non-bulk clients approximates the proportion observed on the live Tor network [MBG+08]. To isolate the improvements due to our proposals, circuit-level prioritization is disabled for this experiment.

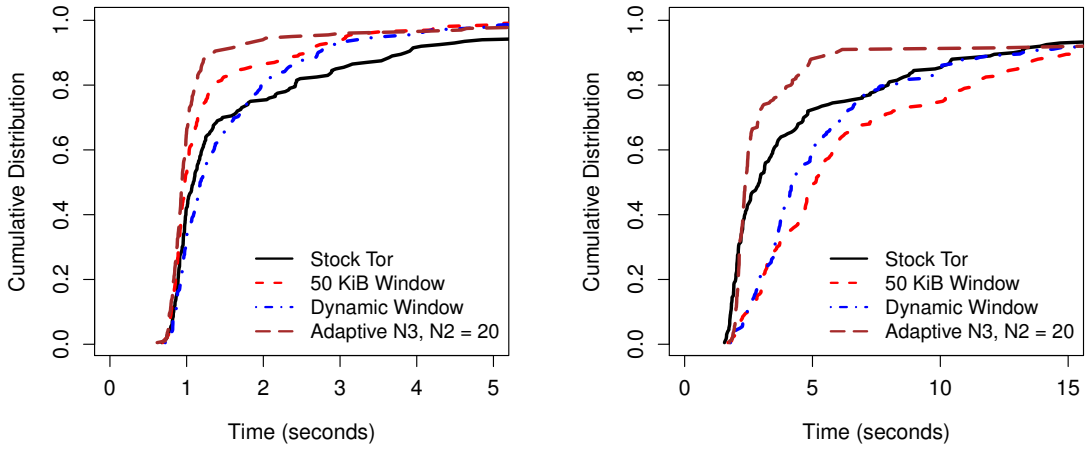
Results. For web clients, Figure 6.10(a) shows that both the 50 KiB fixed and dynamic windows still offer improved time-to-first-byte. However, both algorithms perform worse than stock Tor in terms of overall download time, as shown in Figure 6.10(b). Because smaller windows provide less throughput than larger windows when there is no bottleneck, non-bottlenecked circuits are under-utilized.

N23 with the adaptive N3 algorithm, in contrast, has the ability to react to congestion quickly by reducing routers' queue lengths, causing back pressure to build up. Consequently, our results indicate that N23 offers an improvement in both time-to-first-byte *and* overall download time. This experiment again highlights the potential negative impact of 50 KiB and small dynamic windows, since even in a larger network with a realistic traffic load, smaller windows offer worse performance for typical delay-sensitive web requests relative to Tor's current window size. Thus, to achieve maximal improvements, we suggest that Tor adopt N23 congestion and flow control.

6.4.3 N23 Experiments

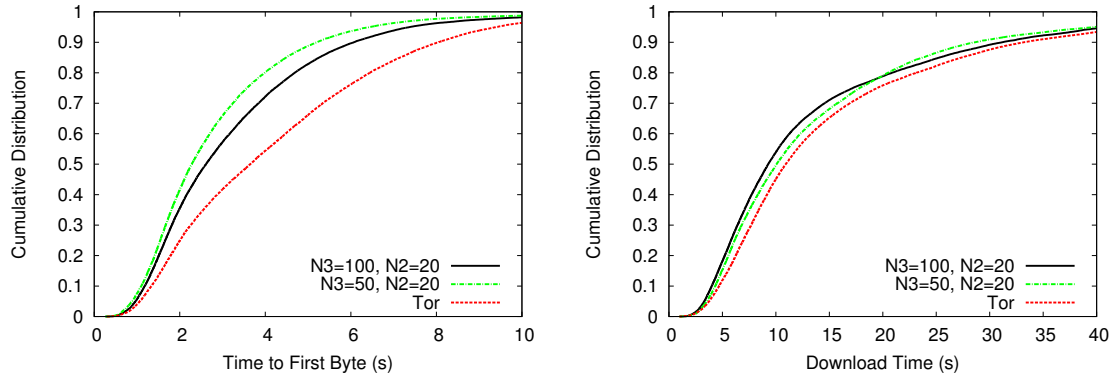
Now that we have established that our window-based approaches can negatively impact the download time of clients, we next focus on more N23 experiments for the remainder of the chapter. We test our N23 algorithm using the larger and more realistic network topology we presented and verified in Section 5.5.1. Recall that this network topology, in which the web to bulk client ratio is 19:1, has been recently proposed to model the performance of the live Tor network, and we have shown in Section 5.5.1 that we indeed obtained a good approximation or the live network.

Figures 6.11(a) and 6.11(b) present our performance results using stock Tor and N23. For N23, we fixed the N2 value to 20, and experimented with two values of N3, which are 50 and 100. In Figure 6.11(a), the time-to-first-byte drops from 3.6 seconds for the stock Tor clients to



(a) Web client's time-to-first-byte (b) Web client's download time

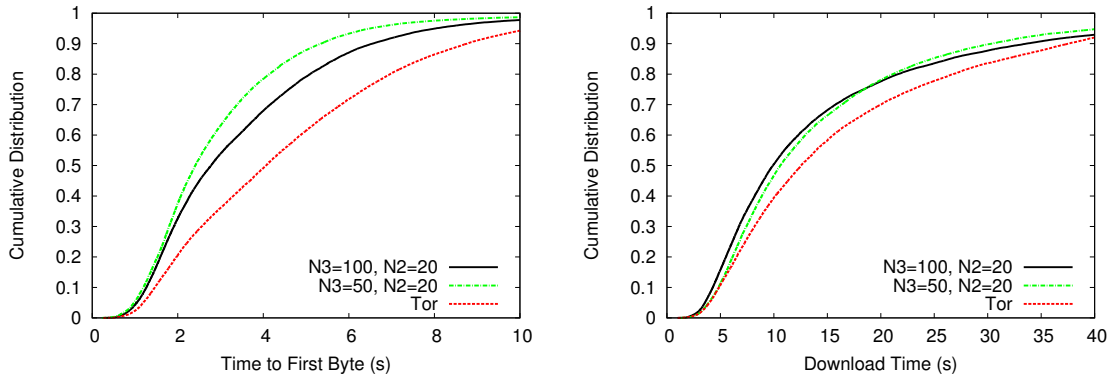
Figure 6.10: Performance results for large-scale experiments



(a) Web client's time-to-first-byte (b) Web client's download time

Figure 6.11: Performance results for N23 large-scale experiments using the network topology presented in 5.5.1.

2.6 seconds when N3 is set to 100 and 2.2 seconds when N3 is decreased to 50. Therefore, when N3 is 100, the improvement is 27%, but decreasing N3 to 50, the improvements increase to 38% at the median.



(a) Web client's time-to-first-byte

(b) Web client's download time

Figure 6.12: Performance results for N23 large-scale experiments using the network topology presented in 5.5.1 with a higher traffic load.

However, increasing the circuit queue size by increasing N3 to 100 slightly lowers the benefits observed in download time. For instance, as shown in Figure 6.11(b), when N3 is set to 100, the median download time is 9.2 seconds, but when N3 is reduced to 50, the median download time increases slighyly to 10 seconds, which is also slightly faster than the Tor client which completes the download in 10.9 seconds.

The benefits of N23 become more visible as we increase the traffic load in the network. Figures 6.12(a) and 6.12(b) compare the network performance for N23 and stock Tor when the bulk to web client ratio is reduced to 9:1. As can be seen in Figure 6.12(a), the time-to-first-byte for the stock Tor client is approximately 4 seconds at the median. When N23 is used with N3 set to 50, the time-to-first-byte is reduced to 2.4 seconds at the median, which translates to an improvement of 40%. Increasing N3 to 100 also yields response time improvements of 32.5%. Download times are improved by 16% at the median when N3 is set to 50 as shown in Figure 6.12(b) compared to Tor. Finally, increasing N3 to 100 increases the download time improvements to approximately 20.8% at the median.

6.5 Discussion

Having empirically evaluated our proposed congestion and flow control approaches, we next discuss a variety of open issues.

6.5.1 Incremental Deployment

In order for our proposed congestion and flow control mechanisms to be practical and easily deployable on the live Tor network, it is important that any modifications to Tor's router infrastructure be incrementally deployable. Any solutions based on Tor's existing window-based flow control require upgrades only to the exit routers; thus they can be slowly deployed as router operators upgrade. N23 may also be deployed incrementally, however, clients may not see substantial performance benefits until a large fraction of the routers have upgraded.

6.5.2 Anonymity Implications

A key question to answer is whether improving Tor's performance and reducing congestion enables any attack that was not previously possible. It is well known that Tor is vulnerable to congestion attacks wherein an attacker constructs circuits through a number of different routers, floods them with traffic, and observes if there is an increase in latency on a target circuit, which would indicate a shared router on both paths [MD05]. More recent work has suggested a solution that would mitigate bandwidth amplification variants of this attack, but not the shared router inference part of the attack [EDG09]. We believe that by reducing congestion (and specifically, by bounding queue lengths), our proposed techniques may increase the difficulty of mounting congestion attacks.

However, if only a fraction of the routers upgrade to our proposals and if clients only choose routers that support the new flow control, then an adversary may be able to narrow down the set of potential routers that a client is using. Thus, it is important to deploy any new flow control technique after a large fraction of the network has upgraded. Such a deployment can be controlled by setting a flag in the authoritative directory servers' consensus document, indicating that it is safe for clients to use the new flow control.

Another well-studied class of attack is end-to-end traffic correlation. Such attacks endeavor to link a client with its destination when the entry and exit points are compromised, and these attacks have been shown to be highly accurate [SS03, ØS06, SW06, BMG+07, MZ07]. Reducing latency might improve this attack; however, Tor is already highly vulnerable, so there is little possibility for additional risk.

Finally, previous work has shown that round-trip time (RTT) can be used as a side channel to infer a possible set of client locations [HVCT07]. By decreasing the variance in latency, we might expose more accurate RTT measurements, thus improving the effectiveness of this attack. However, reducing congestion does not enable a new attack, but rather may potentially increase the effectiveness of a known attack.

6.6 Conclusion

We seek to improve Tor's performance by reducing unnecessary delays due to poor flow control and excessive queuing at intermediate routers. To this end, we have proposed two broad classes of congestion and flow control. First, we tune Tor's existing circuit windows to effectively reduce the amount of data in flight. However, our experiments indicate that while window-based solutions do reduce queuing delays, they tend to suffer from poor flow control, under-utilizing the available bandwidth, and consequently, smaller windows provide slower downloads than unmodified Tor.

To solve this problem, we offer a fresh approach to congestion and flow control in Tor by designing, implementing, and experimentally evaluating a per-link congestion and flow control algorithm from ATM networks. Our experiments indicate that this approach offers the promise of faster web page response times and faster overall web page downloads.

Chapter 7

Conclusion

The goal of this thesis is to enhance the performance of low-latency anonymous communication networks without affecting their anonymity properties. As an experimentation platform, we use Tor, the most widely deployed anonymity network today. To achieve our goals, we focus on the following four main sources of poor performance:

- *Lack of resources.* We observe that since Tor routers are volunteer-operated, they provide limited bandwidth and CPU resources. This problem specifically manifests itself when clients have to use resource-deprived relays such as bridges. To solve this problem, we propose Conflux, a congestion-aware traffic splitting scheme that builds multipath circuits for clients. Conflux dynamically sends on each circuit its desired load by continuously sensing the congestion state of each circuit in the multipath. Our experiments reveal that Conflux can provide significant performance benefits for clients using bridges, and for streaming video clients.
- *Poor quality of service.* One problem in Tor is that the use of greedy applications, such as BitTorrent, can consume a great fraction of the network bandwidth, which can impact the experience of interactive application users. To address this problem, we introduce DiffTor, a machine-learning-based approach that classifies Tor's encrypted circuits in real time. By classifying circuits to three broad application classes, which are interactive, bulk and streaming, relays can perform QoS mapping to assign each traffic class the requirements it needs. For example, interactive applications can receive higher responsiveness, while bulk applications can get throttled or get higher throughput according to the QoS policy defined. Our experiments show that we are able to classify Tor's circuits to a very high accuracy and our live network performance experiments confirm the substantial performance benefits that can be obtained using this approach.

- *Poor transport design.* Because Tor multiplexes several circuits in the same TCP connection between any two routers, TCP congestion control is applied unfairly on all circuits, which can again impact the performance of light interactive circuits. Although this problem has been addressed in previous transport proposals, those proposals face several deployment challenges. To solve this problem, we introduce PCTCP, a novel anonymous communication transport design where every circuit uses a separate TCP connection. PCTCP uses IPsec to protect the network layer from traffic analysis. Our implementation has an easier road to deployment and our extensive experiments show key performance improvements for clients using PCTCP over Tor.
- *Lack of congestion control.* One major problem in Tor that affects its performance is that it is not congestion controlled. That is, intermediate routers can not protect themselves or react in case they experience congestion. Tor only implements an end-to-end window-based flow control algorithm which throttles the amount of traffic that can travel through a circuit. To solve this problem, we implement a credit-based hop-by-hop flow control algorithm, called N23, which allows every router to cap its queues and react to congestion. Our experimental evaluation shows significant performance enhancements when N23 is used.

In conclusion, we have ascertained in this thesis that it is possible to improve the performance of anonymity networks, exemplified by Tor, while maintaining the anonymity they provide to users. In addition to the substantial performance benefits that can be obtained using our proposed improvements, our techniques are practical and are easily deployable. We hope that our techniques would be useful and applicable for the Tor network and in the design of future low-latency anonymous communication networks.

References

[ABG+11] Mashael AlSabah, Kevin Bauer, Ian Goldberg, Dirk Grunwald, Damon McCoy, Stefan Savage, and Geoffrey M. Voelker. DefenestraTor: Throwing out Windows in Tor. In *11th Privacy Enhancing Technologies Symposium*, pages 134–154, July 2011.

[ABG12] Mashael AlSabah, Kevin Bauer, and Ian Goldberg. Enhancing Tor's performance using real-time traffic classification. In *Proceedings of the 19th ACM conference on Computer and Communications Security*, CCS '12, pages 73–84, New York, NY, USA, 2012. ACM.

[Ale12] Alexa: The Web Information Company. http://www.alexa.com/topsites, 2012. Accessed February 2012.

[All10] Mark Allman. Internet Draft: Initial Congestion Window Specification. http://tools.ietf.org/html/draft-allman-tcpm-bump-initcwnd-00, November 2010. Accessed April 2013.

[AMG07] Tom Auld, Andrew W. Moore, and Stephen F. Gull. Bayesian Neural Networks for Internet Traffic Classification. *IEEE Transactions on Neural Networks*, 18(1):223–239, January 2007.

[Ano] The Anonymizer. http://www.anonymizer.com. Accessed February 2013.

[APB09] Mark Allman, Vern Paxson, and Ethan Blanton. RFC 5681: TCP Congestion Control. http://tools.ietf.org/html/rfc5681, September 2009. Accessed April 2013.

[App12] Jacob Appelbaum. Tor and NAT devices: increasing bridge & relay reachability or, enabling the use of NAT-PMP and UPnP by default. Technical Report 2012-08-001, The Tor Project, August 2012. Accessed April 2013.

[AYM12] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. LASTor: A Low-Latency AS-Aware Tor Client. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 476–490, Washington, DC, USA, 2012. IEEE Computer Society.

[BB07] BBC News. Data disaster: Your queries answered. http://news.bbc.co.uk/2/hi/business/7105592.stm, November 2007. Accessed January 2013.

[BDHR95] Torsten Braun, Christophe Diot, Anna Hoglander, and Vincent Roca. An Experimental User Level Implementation of TCP. Technical Report RR-2650, INRIA, September 1995.

[BDMT07] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of Service or Denial of Security? How Attacks on Reliability can Compromise Anonymity. In *Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS '07*, pages 92–102, October 2007.

[BMC+11] Stevens Le Blond, Pere Manils, Abdelberi Chaabane, Mohamed Ali Kaafar, Claude Castelluccia, Arnaud Legout, and Walid Dabbous. One Bad Apple Spoils the Bunch: Exploiting P2P Applications to Trace and Profile Tor Users. In *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats (LEET)*, March 2011.

[BMG+07] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-Resource Routing Attacks against Tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)*, pages 11–20, October 2007.

[BOP94] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, pages 24–35, New York, NY, USA, 1994. ACM.

[Bro02] Zach Brown. Cebolla: Pragmatic IP Anonymity. http://www.cypherspace.org/cebolla/cebolla.pdf, June 2002. Accessed February 2013.

[BSG00] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom Systems 2.0 Architecture. www.cypherspace.org/adam/pubs/freedom2-mail.pdf, December 2000. Accessed February 2013.

[BSMG11] Kevin Bauer, Micah Sherr, Damon McCoy, and Dirk Grunwald. ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation. In *Proceedings of the 4th USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, pages 51–59, August 2011.

[BTA+06] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic Classification on the Fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, April 2006.

[CB12] CBC News. Cyberbullying-linked Suicides rising, study says. http://www.cbc.ca/news/technology/story/2012/10/19/cyberbullying-suicide-study.html, October 2012. Accessed January 2013.

[CDC11] Jerry Chu, Nandita Dukkipati, and Yuchung Cheng. Internet Draft: Increasing TCP's Initial Window. http://tools.ietf.org/id/draft-ietf-tcpm-initcwnd-01.txt, April 2011. Accessed April 2013.

[CFG09] Dario Catalano, Dario Fiore, and Rosario Gennaro. Certificateless Onion Routing. In *Proceedings of the 16th ACM conference on Computer and Communications Security*, CCS '09, pages 151–160, New York, NY, USA, 2009. ACM.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2):84–90, February 1981.

[Ch] The Chromium Projects. SPDY: An Experimental Protocol for a Faster Web. http://dev.chromium.org/spdy/spdy-whitepaper. Accessed April 2013.

[CMK10] Abdelberi Chaabane, Pere Manils, and Mohamed Ali Kaafar. Digging into Anonymous Traffic: A Deep Analysis of the Tor Anonymizing Network. In *Proceedings of the 4th International Conference on Network and System Security (NSS)*, pages 167–174, September 2010.

[Coh11] Bram Cohen. BitTorrent Protocol Specification. http://wiki.theory.org/BitTorrentSpecification, June 2011. Accessed August 2011.

[CP08] Fallon Chen and Mike Perry. Improving Tor Path Selection. https://gitweb.torproject.org/torspec.git/blob_plain/HEAD:/proposals/151-path-selection-improvements.txt, July 2008. Accessed April 2013.

[DB13] Anupam Das and Nikita Borisov. Securing Tor Tunnels under the Selective-DoS Attack. In *Proceedings of Financial Cryptography and Data Security*, February 2013.

[DCRS12] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 332–346, May 2012.

[DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, May 2003.

[Din09] Roger Dingledine. Prop 168: Reduce default circuit window. https://gitweb.torproject.org/torspec.git/blob_plain/HEAD:/proposals/168-reduce-circwindow.txt, August 2009. Accessed April 2013.

[Din10] Roger Dingledine. Bittorrent over Tor isn't a good idea. https://blog.torproject.org/blog/bittorrent-over-tor-isnt-good-idea, April 2010. Accessed April 2013.

[Din11] Roger Dingledine. Tor and Circumvention: Lessons Learned. In *Proceedings of the 31st Annual Conference on Advances in Cryptology (CRYPTO)*, pages 485–486, August 2011.

[DM06] Roger Dingledine and Nick Mathewson. Anonymity Loves Company: Usability and the Network Effect. In *Workshop on the Economics of Information Security*, pages 547–559, June 2006.

[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, March 2009. Accessed April 2013.

[DM13] Roger Dingledine and Nick Mathewson. Tor protocol specification. https://gitweb.torproject.org/torspec.git/blob/HEAD:/tor-spec.txt, April 2013. Accessed April 2013.

[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, August 2004.

[DR08] Tim Dierks and Eric Rescorla. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2. http://www.ietf.org/rfc/rfc5246.txt, August 2008. Accessed April 2013.

[DSR+10] Prithula Dhungel, Moritz Steiner, Ivinko Rimac, Volker Hilt, and Keith W. Ross. Waiting for Anonymity: Understanding Delays in the Tor Overlay. In *Peer-to-Peer Computing*, pages 1–4, 2010.

[EBA+12] Tariq Elahi, Kevin Bauer, Mashael AlSabah, Roger Dingledine, and Ian Goldberg. Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2012)*. ACM, October 2012.

[EBR03] James P. Early, Carla E. Brodley, and Catherine Rosenberg. Behavioral Authentication of Server Flows. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC)*, pages 46–56, December 2003.

[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, Berkeley, CA, USA, 2009. USENIX Association.

[EM95] Aled Edwards and Steve Muir. Experiences implementing a high performance TCP in user-space. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '95, pages 196–205. ACM, 1995.

[ES09] Matthew Edman and Paul F. Syverson. AS-awareness in Tor Path Selection. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 380–389, 2009.

[FGG97] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian Network Classifiers. *Mach. Learn.*, 29(2-3):131–163, November/December 1997.

[FJS10] Joan Feigenbaum, Aaron Johnson, and Paul Syverson. Preventing Active Timing Attacks in Low-Latency Anonymous Communication. In *Proceedings of the 10th Privacy Enhancing Technologies Symposium*, pages 166–183, 2010.

[GALM08] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Characterizing User Sessions on YouTube. In *Proceedings of the 15th IEEE Multimedia Computing and Networking (MMCN)*, January 2008.

[Gam01] Joao Gama. Functional trees for classification. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 147–154, Washington, DC, USA, 2001. IEEE Computer Society.

[GH12] Deepika Gopal and Nadia Heninger. Torchestra: Reducing Interactive Traffic Delays over Tor. In *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*, WPES '12, pages 31–42, New York, NY, USA, 2012. ACM.

[GKL+09] Thomer M. Gil, Frans Kaashoek, Jinyang Li, Robert Morris, and Jeremy Stribling. King Data Set. http://pdos.csail.mit.edu/p2psim/kingdata, November 2009. Accessed August 2011.

[Gol10] Ian Goldberg. Prop 174: Optimistic Data for Tor: Server Side. https://trac.torproject.org/projects/tor/ticket/1795, August 2010. Accessed April 2013.

[Gol11] Ian Goldberg. Prop 181: Optimistic Data for Tor: Client Side. https://trac.torproject.org/projects/tor/ticket/3564, July 2011. Accessed April 2013.

[Gre06] Larry Greenemeier. VA Secretary Comes Under Fire At House And Senate Data Theft Hearings. http://www.informationweek.com/va-secretary-comes-under-fire-at-house-a/188500312, May 2006. Accessed January 2013.

[Gre08] Larry Greenemeier. Security Breach: Feds Lose Laptop Containing Sensitive Data – Again. http://www.scientificamerican.com/article.cfm?id=security-breach-lost-laptop, 2008. Accessed January 2013.

[GSU11] Ian Goldberg, Douglas Stebila, and Berkant Ustaoglu. Anonymity and One-way Authentication in Key Exchange Protocols. University of Waterloo Technical Report CACR 2011-05, May 2011.

[HCJS03] Félix Hernández-Campos, Kevin Jeffay, and F. Donelson Smith. Tracking the Evolution of Web Traffic: 1995-2003. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS)*, pages 16–25, 2003.

[Hed91] Charles L. Hedrick. An Introduction to IGRP. http://www.cisco.com/image/gif/paws/26825/5.pdf, August 1991. Accessed April 2013.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[HSH+06] Huaizhong Han, Srinivas Shakkottai, C. V. Hollot, R. Srikant, and Don Towsley. Multi-path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet. *IEEE/ACM Trans. Netw.*, 14(6):1260–1271, December 2006.

[HVCT07] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? In *Proceedings of the 14th ACM conference on Computer and Communications Security*, CCS '07, October 2007.

[HWF09] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In *Proceedings of the ACM Cloud Computing Security Workshop (CCSW)*, pages 31–42, November 2009.

[iMa12] iMacros for Firefox. https://addons.mozilla.org/en-US/firefox/addon/imacros-for-firefox/, 2012. Accessed February 2012.

[iPl] iPlane: Data. http://iplane.cs.washington.edu/data/data.html. Accessed February 2013.

[Jai95] Raj Jain. Congestion Control and Traffic Management in ATM Networks: Recent Advances and a Survey. *Computer Networks and ISDN Systems*, 28:1723–1738, 1995.

[JBHD12] Rob Jansen, Kevin Bauer, Nicholas Hopper, and Roger Dingledine. Methodically Modeling the Tor Network. In *Proceedings of the USENIX Workshop on Cyber Security Experimentation and Test (CSET 2012)*, August 2012.

[JH12] Rob Jansen and Nicholas Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Proceedings of the 19th Network and Distributed Security Symposium*, February 2012.

[JHK10] Rob Jansen, Nicholas Hopper, and Yongdae Kim. Recruiting New Tor Relays with BRAIDS. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 319–328, New York, NY, USA, 2010. ACM.

[JSH12] Rob Jansen, Paul Syverson, and Nicholas Hopper. Throttling Tor Bandwidth Parasites. In *Proceedings of the 21st USENIX Security Symposium*, Berkeley, CA, USA, 2012. USENIX Association.

[KA98] Stephen Kent and Randall Atkinson. RFC 2401: Security Architecture for the Internet Protocol. http://tools.ietf.org/html/rfc2401, November 1998. Accessed February 2013.

[Kar06] Teknomo Kardi. Recursive Average and Variance. http://people.revoledu.com/kardi/tutorial/RecursiveStatistic/index.html, 2006. Accessed January, 2012.

[KBC94] H. T. Kung, Trevor Blackwell, and Alan Chapman. Credit-based flow control for ATM networks: credit update protocol, adaptive credit allocation and statistical multiplexing. *ACM SIGCOMM Computer Communication Review*, 24:101–114, October 1994.

[KBC05] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Trans. Dependable Secur. Comput.*, 2(2):93–108, April 2005.

[KBC08] C. Kiraly, G. Bianchi, and R. Lo Cigno. Solving Performance Issues in Anonymization Overlays with a L3 Approach. University of Trento Information Engineering and Computer Science Department Technical Report DISI-08-041, September 2008.

[KCF+08] Hyunchul Kim, KC Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *Proceedings of the ACM CoNEXT Conference*, pages 11:1–11:12, December 2008.

[KHSV05] Tero Kivinen, Ari Huttunen, Brian Swander, and Victor Volpe. RFC 53947: Negotiation of NAT-Traversal in the IKE. http://www.ietf.org/rfc/rfc3947.txt, January 2005. Accessed April 2013.

[Kin12] Andrew King. Average Web Page Size Septuples Since 2003. Website Optimization, LLC. http://www.websiteoptimization.com/speed/tweak/average-web-page, November 2012. Accessed February 14, 2013.

[KZG07] Aniket Kate, Greg Zaverucha, and Ian Goldberg. Pairing-based Onion Routing. In *Proceedings of the 7th Privacy Enhancing Technologies Symposium*, PETS '07, pages 95–112, Berlin, Heidelberg, 2007. Springer-Verlag.

[Lew10] Andrew Lewman. China Blocking Tor: Round Two. https://blog.torproject.org/blog/china-blocking-tor-round-two, March 2010. Accessed August 2011.

[LHF05] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic Model Trees. *Mach. Learn.*, 59(1-2):161–205, May 2005.

[LL06] Marc Liberatore and Brian Neil Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*, pages 255–263, October 2006.

[LLY+12] Zhen Ling, Junzhou Luo, Wei Yu, Ming Yang, and Xinwen Fu. Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery. In *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM)*, March 2012.

[Loe10] Karsten Loesing. Safely collecting data to estimate the number of Tor users. https://lists.torproject.org/pipermail/tor-dev/2010-August/000467.html, August 2010. Accessed April 2012.

[LPW+07] Olaf Landsiedel, Alexis Pimenidis, Klaus Wehrle, Heiko Niedermayer, and Georg Carle. Dynamic Multipath Onion Routing in Anonymous Peer-to-Peer Overlay Networks. In *IEEE Global Telecommunications Conference, 2007*, pages 64–69, Nov. 2007.

[LRWW04] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew K. Wright. Timing Attacks in Low-Latency Mix-Based Systems. In *Proceedings of Financial Cryptography*, pages 251–265, February 2004.

[MBG+08] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining Light in Dark Places: Understanding the Tor Network. In *Proceedings of the 8th Privacy Enhancing Technologies Symposium*, pages 63–76, July 2008.

[MD05] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.

[MFPA09] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On Dominant Characteristics of Residential Broadband Internet Traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 90–102, November 2009.

[MKJ+11] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, CCS '11, pages 215–226, New York, NY, USA, 2011. ACM.

[MOT+11] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *Proceedings of the 20th USENIX Security Symposium*, August 2011.

[Moy98] John Moy. RFC 2328: OSPF Version 2. http://tools.ietf.org/html/rfc2328, April 1998. Accessed February, 2012.

[MTHK09] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable Onion Routing with Torsk. In *Proceedings of the 16th ACM conference on Computer and Communications Security*, CCS '09, pages 590–599, New York, NY, USA, 2009. ACM.

[Mur11] Steven J. Murdoch. Comparison of Tor Datagram Designs. *Tor Project Technical Report*, November 2011.

[MW08] Steven J. Murdoch and Robert N. M. Watson. Metrics for Security and Performance in Low-Latency Anonymity Systems. In *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, pages 115–132, Leuven, Belgium, July 2008.

[MWS11] W. Brad Moore, Chris Wacek, and Micah Sherr. Exploring the Potential Benefits of Expanded Rate Limiting in Tor: Slow and Steady Wins the Race with Tortoise. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC)*, pages 207–216, December 2011.

[MZ05] Andrew W. Moore and Denis Zuev. Internet Traffic Classification using Bayesian Analysis Techniques. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 50–60, June 2005.

[MZ07] Steven J. Murdoch and Piotr Zieliński. Sampled traffic analysis by Internet-exchange-level adversaries. In *Proceedings of Privacy Enhancing Technologies Workshop (PET 2007)*, June 2007.

[Nah04] Fiona Fui-Hoon Nah. A Study on Tolerable Waiting Time: How long are Web Users Willing to Wait? *Behaviour Information Technology*, 23(3):153–163, 2004.

[NDW10] Tsuen-Wan "Johnny" Ngan, Roger Dingledine, and Dan S. Wallach. Building Incentives into Tor. In *Proceedings of Financial Cryptography and Data Security*, pages 238–256, January 2010.

[Oo] Ookla. Net Index by Ookla — Source Data. http://www.netindex.com/source-data. Accessed on January 27, 2012.

[Ope13] OpenSwan. https://www.openswan.org/projects/openswan/, April 2013. Accessed April 2013.

[ØS06] Lasse Øverlier and Paul Syverson. Locating Hidden Servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 100–114, May 2006.

[ØS07] Lasse Øverlier and Paul Syverson. Improving efficiency and simplicity of Tor circuit establishment and hidden services. In *Proceedings of the 7th Privacy Enhancing Technologies Symposium*, PETS '07, pages 134–152, Berlin, Heidelberg, 2007. Springer-Verlag.

[PFTK98] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, Technologies, Architectures, and Protocols for*

Computer Communication, SIGCOMM '98, pages 303–314, New York, NY, USA, 1998. ACM.

[PKK08] Michael Piatek, Tadayoshi Kohno, and Arvind Krishnamurthy. Challenges and Directions for Monitoring P2P File Sharing Networks-or: Why My Printer Received a DMCA Takedown Notice. In *Proceedings of the 3rd Conference on Hot Topics in Security*, pages 12:1–12:7, July 2008.

[PNZE11] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 103–114, October 2011.

[Ram12] Sreeram Ramachandran. Web Metrics: Size and Number of Resources. https://code.google.com/speed/articles/web-metrics.html, March 2012. Accessed August 2011.

[Rea08] Joel Reardon. Improving Tor using a TCP-over-DTLS Tunnel. Unversity of Waterloo Master's Thesis, October 2008.

[RG09] Joel Reardon and Ian Goldberg. Improving Tor Using a TCP-over-DTLS Tunnel. In *Proceedings of the 18th USENIX Security Symposium*, August 2009.

[RLL+11] Ashwin Rao, Arnaud Legout, Yeon-sup Lim, Don Towsley, Chadi Barakat, and Walid Dabbous. Network Characteristics of Video Streaming Traffic. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 25:1–25:12, New York, NY, USA, 2011. ACM.

[RSG98] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous Connections and Onion Routing. *Selected Areas in Communications, IEEE Journal on*, 16(4):482 –494, May 1998.

[RSSD04] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 135–148, October 2004.

[Sa11] Sandvine. Sandvine Global Internet Phenomena Report — Fall 2011. http://www.sandvine.com/downloads/documents/Phenomena_1H_2012/Sandvine_Global_Internet_Phenomena_Report_1H_2012.pdf, October 2011. Accessed April 2013.

[SB08] Robin Snader and Nikita Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *Proceedings of the Network and Distributed Security Symposium (NDSS)*, February 2008.

[SBB05] Rob Sherwood, Bobby Bhattacharjee, and Ryan Braud. Misbehaving TCP Receivers can cause Internet-wide Congestion Collapse. In *Proceedings of the 12th ACM conference on Computer and Communications Security*, CCS '05, pages 383–392, New York, NY, USA, 2005. ACM.

[SBL09] Micah Sherr, Matt Blaze, and Boon Thau Loo. Scalable Link-Based Relay Selection for Anonymous Routing. In *PETS '09: Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, pages 73–93, Berlin, Heidelberg, 2009. Springer-Verlag.

[SCWA99] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP Congestion Control with a Misbehaving Receiver. *ACM SIGCOMM Computer Communication Review*, 29:71–78, 1999.

[She12] Andrea Shepard. Build Abstraction Layer Around TLS. https://trac.torproject.org/projects/tor/ticket/6465, July 2012. Accessed February 2013.

[SM05] Andrei Serjantov and Steven J. Murdoch. Message Splitting Against the Partial Adversary. In *Proceedings of Privacy Enhancing Technologies Workshop*, pages 26–39, May 2005.

[Sna10] Robin Snader. *Path Selection for Performance- and Security-Improved Onion Routing*. PhD thesis, University of Illinois at Urbana-Champaign, 2010.

[SS03] Andrei Serjantov and Peter Sewell. Passive Attack Analysis for Connection-Based Anonymity Systems. In *Proceedings of ESORICS'03*, pages 116–131, October 2003.

[SSGC05] Craig Shue, Youngsang Shin, Minaxi Gupta, and Jong Youl Choi. Analysis of IPSec overheads for VPN servers. In *Proceedings of the First international Conference on Secure Network Protocols*, NPSEC '05, pages 25–30, Washington, DC, USA, 2005. IEEE Computer Society.

[STRL00] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an Analysis of Onion Routing Security. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114, July 2000.

[SW06] Vitaly Shmatikov and Ming-Hsui Wang. Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses. In *Proceedings of ESORICS'06*, pages 18–33, September 2006.

[TG10] Can Tang and Ian Goldberg. An Improved Algorithm for Tor Circuit Scheduling. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, pages 329–339, October 2010.

[THK09] Andrew Tran, Nicholas Hopper, and Yongdae Kim. Hashing It out in Public: Common Failure Modes of DHT-based Anonymitys Schemes. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES'09)*, pages 71–80, November 2009.

[To09] The Tor Project. Tor Bridges Specification. https://gitweb.torproject.org/torspec.git/blob_plain/HEAD:/bridges-spec.txt, May 2009. Accessed August 2011.

[To12a] The Tor Project. Tor Metrics Portal: Bandwidth History by Relay Flags. https://metrics.torproject.org/network.html?graph=bwhist-flags&start=2012-07-01&end=2012-07-02&dpi=72#bwhist-flags, July 2012. Accessed April 2013.

[To12b] The Tor Project. Tor Metrics Portal: Data. https://metrics.torproject.org/data.html#performance, July 2012. Accessed October 2012.

[To12c] The Tor Project. Tor Metrics Portal: Network. http://metrics.torproject.org/network.html?graph=networksize&start=2012-01-01&end=2012-01-31&dpi=72#networksize, July 2012. Accessed April 2013.

[To12d] The Tor Project. Tor Metrics Portal: Users. http://metrics.torproject.org/users.html, July 2012. Accessed April 2013.

[To13] The Tor Project. Codename: Torouter. https://trac.torproject.org/projects/tor/wiki/doc/Torouter, January 2013. Accessed February 2013.

[TS11] Florian Tschorsch and Björn Scheurmann. Tor is unfair — And what to do about it. In *Proceedings of the 36th IEEE Conference on Local Computer Networks (LCN)*, pages 432–440, October 2011.

[TS12] Florian Tschorsch and Björn Scheurmann. How (not) to Build a Transport Layer for Anonymity Overlays. In *Proceedings of the ACM Sigmetrics/Performance Workshop on Privacy and Anonymity for the Digital Economy*, June 2012.

[Vie08] Camilo Viecco. UDP-OR: A Fair Onion Transport Design. http://www.petsymposium.org/2008/hotpets/udp-tor.pdf, July 2008. Accessed February 2013.

[Vuz12] Vuze: The Most Powerful BitTorrent App on Earth. http://www.vuze.com, 2012. Accessed February 2012.

[VYW+02] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and Accuracy in a Large-scale Network Emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, December 2002.

[WALS04] Matthew K. Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems. *ACM Transactions on Information and System Security (TISSEC)*, 7(4):489–522, 2004.

[WBFG12] Tao Wang, Kevin Bauer, Clara Forero, and Ian Goldberg. Congestion-aware Path Selection for Tor. In *Proceedings of Financial Cryptography and Data Security (FC '12)*, February 2012.

[WC92] Zheng Wang and Jon Crowcroft. Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm. *ACM SIGCOMM Computer Communication Review*, 22:9–16, April 1992.

[WCM09] Charles Wright, Scott Coull, and Fabian Monrose. Traffic Morphing: An Efficient Defense against Statistical Traffic Analysis. In *Proceedings of the 16th Network and Distributed Security Symposium*, February 2009.

[WTBS13] Chris Wacek, Henry Tan, Kevin Bauer, and Micah Sherr. An Empirical Evaluation of Relay Selection in Tor. In *Proceedings of the Network and Distributed Security Symposium (NDSS)*, February 2013.

[XMH11] Xueyang Xu, Zhuoqing Morley Mao, and J. Alex Halderman. Internet Censorship in China: Where Does the Filtering Occur? In *PAM*, pages 133–142, 2011.

[ZM05] Denis Zuev and Andrew W. Moore. Traffic Classification using a Statistical Approach. In *Proceedings of the 6th International Conference on Passive and Active Network Measurement*, PAM '05, pages 321–324, 2005.