DOC'Tor: Defending and Opening Communication on Tor

A Dissertation presented

by

Rishab Nithyanand

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

May 2017
Stony Brook University
The Graduate School

Rishab Nithyanand

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation

Rob Johnson (Advisor), Research Assistant Professor
Department of Computer Science
Stony Brook University

Phillipa Gill (Advisor), Research Assistant Professor
Department of Computer Science
Stony Brook University

Nick Nikiforakis, Assistant Professor
Department of Computer Science
Stony Brook University

Vern Paxson, Professor
Department of Electrical Engineering and Computer Sciences
University of California - Berkeley

This dissertation is accepted by the Graduate School

Charles Taber
Dean of the Graduate School
Abstract of the Dissertation

DOCTor: Defending and Opening Communication on Tor

by

Rishab Nithyanand

Doctor of Philosophy

in

Computer Science

Stony Brook University

2017

The Tor anonymity network relies heavily on volunteer-owned and operated resources to service millions of users each day. Consequently, it needs to manage these resources efficiently while addressing challenges to its robustness and utility. Many challenges faced by Tor arise from a deficit of trust in three entities—relay operators responsible for managing the relays through which Tor traffic flows, Autonomous Systems (ASes) that own the networks in which relays operate, and users interacting with the Internet via the Tor network. Specifically, relay operators may use traffic flow characteristics to identify the content accessed by a Tor user, ASes may place themselves in positions to exactly identify Tor users and the servers being accessed by them, network-level adversaries (e.g., ASes on behalf of restrictive governments) may reduce the utility of the network by identifying and blocking Tor traffic, and users may misuse the anonymity provided by the network.

In this dissertation, we address each of these threats. In particular, we present (1) traffic flow modification strategies to counter the threat from relay-level and eavesdropping adversaries, (2) relay selection strategies that utilize the state-of-the-art in Internet measurement to mitigate the threat of deanonymization by network-level adversaries, (3) an extensible covert-channel construction framework that addresses the threat of blocking by network-level adversaries by reversing the resource imbalance in the arms race between censors and circumvention tool developers, and (4) measurements that quantify server-side discrimination faced by legitimate Tor users as a conse-
quence of abusive behavior from malicious users of the network.

At a high-level, this dissertation presents theoretically and empirically derived ideas for increasing the robustness of any network. The proposed flow modification strategies demonstrate how provably secure traffic correlation defenses can be bootstrapped even with limited bandwidth resources. Our relay selection strategies show how to prevent traffic correlation attacks by utilizing network measurement research to route around adversaries and without requiring changes to the network infrastructure. Our covert-channel framework illustrates how appropriate protocol selection can make blocking of communication more expensive for censors. Finally, our measurements of server-side discrimination show one of the costs of anonymous communication in a public network.
Dedicated to my parents and teachers for demonstrating

the value of goodness, discipline, and knowledge.
5.2 Background and Related Work ........................................... 95
5.3 Our Deployed Exits ...................................................... 96
5.4 Synthetic Discrimination Measurements .......................... 97
5.5 User-driven Discrimination Measurements ....................... 105
5.6 Conclusions .............................................................. 107

6 Dataset and Software Releases .......................................... 111

Bibliography ........................................................................... 113
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Growth of the Tor network since 2010</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>CS-BuFLO: Rate adaptation</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>CS-BuFLO: Early termination</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>CS-BuFLO: Security evaluation</td>
<td>31</td>
</tr>
<tr>
<td>2.4</td>
<td>CS-BuFLO: Bandwidth overhead evaluation</td>
<td>32</td>
</tr>
<tr>
<td>2.5</td>
<td>Theoretical lower-bounds: Comparing existing defense overheads</td>
<td>33</td>
</tr>
<tr>
<td>2.6</td>
<td>Glove: Security and overhead evaluation</td>
<td>38</td>
</tr>
<tr>
<td>2.7</td>
<td>Glove: Impact of configuration parameters</td>
<td>38</td>
</tr>
<tr>
<td>3.1</td>
<td>Network-level correlation attack pre-requisites</td>
<td>42</td>
</tr>
<tr>
<td>3.2</td>
<td>Network-level path prediction: Accuracy of algorithmic simulations</td>
<td>46</td>
</tr>
<tr>
<td>3.3</td>
<td>Network-level path prediction: Accuracy of PathCache</td>
<td>47</td>
</tr>
<tr>
<td>3.4</td>
<td>PathCache coverage: Breakdown by Tor client location</td>
<td>48</td>
</tr>
<tr>
<td>3.5</td>
<td>Network-level correlation attacks: Current vulnerability (web workload)</td>
<td>49</td>
</tr>
<tr>
<td>3.6</td>
<td>Network-level correlation attacks: Current vulnerability (mixed workload)</td>
<td>50</td>
</tr>
<tr>
<td>3.7</td>
<td>Network-level correlation attacks: Historical vulnerability (web workload)</td>
<td>51</td>
</tr>
<tr>
<td>3.8</td>
<td>Network-level correlation attacks: Historical vulnerability (mixed workload)</td>
<td>51</td>
</tr>
<tr>
<td>3.9</td>
<td>Growth of the Tor network since 2010</td>
<td>52</td>
</tr>
<tr>
<td>3.10</td>
<td>Network-level traffic correlation attacks: Impact of asymmetric routes</td>
<td>53</td>
</tr>
<tr>
<td>3.11</td>
<td>Network-level correlation attacks: Impact of exploiting BGP dynamics</td>
<td>54</td>
</tr>
<tr>
<td>3.12</td>
<td>Network-level correlation attacks: Vulnerability of Tor relays to BGP exploits</td>
<td>54</td>
</tr>
<tr>
<td>3.13</td>
<td>Network-level correlation attacks: Availability of attacker-free options</td>
<td>55</td>
</tr>
<tr>
<td>3.14</td>
<td>Network-level correlation attacks: Fraction of connections with less than 5% safe circuits</td>
<td>55</td>
</tr>
<tr>
<td>3.15</td>
<td>Network-level correlation attacks: Impact of number of guards on vulnerability</td>
<td>56</td>
</tr>
<tr>
<td>3.16</td>
<td>Security evaluation: Cipollino vs. Astoria vs. vanilla Tor</td>
<td>59</td>
</tr>
<tr>
<td>3.17</td>
<td>Cipollino performance tuning: Impact of pre-emptive circuit construction on safe circuit availability</td>
<td>60</td>
</tr>
<tr>
<td>3.18</td>
<td>Cipollino performance tuning: Impact of pre-emptive circuit construction on circuit allocation times</td>
<td>60</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

3.19 Performance evaluation: Cipollino vs. Astoria vs. vanilla Tor. ........................................... 61
3.20 Cipollino security tuning: Impact of pre-emptive circuit construction on number of unique relays utilized in Cipollino circuits. ................................................................. 61
3.21 Load-balancing: Cipollino vs. Astoria vs. vanilla Tor. ........................................................... 62
3.22 Cipollino: Optimizing relay selection in the worst-case. ......................................................... 63
3.23 Cipollino: System architecture. ............................................................................................... 64

4.1 Growth of the real-time strategy game video game genre on the Steam distribution platform. ................................................................. 69
4.2 Castle architecture overview. ................................................................................................. 75
4.3 Castle security evaluation: Impact of parameters on packet size and time distributions. ........ 83
4.4 Castle security evaluation: Impact of parameters on classifier accuracy. ............................. 84
4.5 Castle performance evaluation: Impact of parameters on throughput. ................................ 87
4.6 Castle performance evaluation: Throughput with and without game-specific enhancements. ........................................................................ 89

5.1 Synthetic measurements: Characteristics of selected relays. ................................................. 98
5.2 Synthetic measurements: Availability of selected relays. ..................................................... 99
5.3 Synthetic measurements: Results of pilot study. .................................................................... 100
5.4 Synthetic measurements: Distribution of observed discrimination. ..................................... 101
5.5 Synthetic measurements: Discrimination performed by popular hosting providers. ................ 102
5.6 Synthetic measurements: Impact of relay characteristics on discrimination. ..................... 102
5.7 Synthetic measurements: Impact of relay characteristics on discrimination performed by popular hosting providers. ...................................................... 103
5.8 Synthetic measurements: Impact of service category on discrimination. ............................. 103
5.9 Synthetic measurements: Discrimination over time. .............................................................. 104
5.10 Synthetic measurements: Impact of measurement methodology. ........................................ 105
5.11 User-driven measurements: Fraction of errors observed. ................................................... 107
5.12 Synthetic measurements: Discriminating websites. .............................................................. 110
Acknowledgements

During my years as a student at Stony Brook I’ve learned that the pursuit of the Ph.D. is a character building experience filled with moments of great highs and deep lows. Looking back, I realize that it was not a journey that I could have completed on my own. I’ve been extremely fortunate to have had many wonderful people around me to celebrate and commiserate with.

First, I am very thankful for all the love and support from my parents and sister. There are no words to describe how much their encouragement has meant to me. Their struggle to give me a good education is something I owe lifetimes of gratitude for.

I have also been lucky to have two great advisors – Rob Johnson and Phillipa Gill. I would not be writing this dissertation if it wasn’t for their support. In addition to being thankful for his always brilliant insights and encouraging comments, I will always be grateful to Rob for his faith and willingness to help me through some tough times when I was a struggling second-year. I know of few other students who were lucky enough to have an advisor that guided and encouraged research pursuits in areas as diverse as security, measurement, usability, and algorithms. The freedom I was given to pursue anything I found interesting played a big part in keeping me motivated. As for Phillipa, if I’d known that the chat I had with her in my fourth year would end up being an adrenaline shot for my research development, I would have done it sooner. The time she spent to make my talks better, her intensity during deadline pushes, openness to new ideas, and the opportunities she made available to me along the way were all invaluable. Now, I’m looking forward to joining her at UMass as a post-doc for more fun and productive years. It’s hard to not enjoy research when you’re being backed by people like Rob and Phillipa. It’s also been a pleasure to have had Vern Paxson and Nick Nikiforakis on my committee. Their comments and encouragement during my prelim and defense are things I will remember for a while. I’m also thankful to Vern for hosting me at ICSI.

I definitely need to thank my lab-mates and colleagues who’ve carried me along the way. I’ll always look forward to more discussions about life, research, ethics, philosophy, dogs, and cars (not always disjoint) with Xiang, Jun, Roozbeh, Golnaz, Jiemin, Polina, Moussa, Abbas, Rachee, and Shinyoung. At ICSI it was fun to work with Narseo, Mobin, David, Michael, and Sadia. I’m thankful to the folks at Qualcomm Research and Sandia National Labs – Rosario, Olivier, Anand,
John, and Tom – for action packed and memorable summers in California. Along the way, I’ve also been lucky to have had the opportunity to learn from and work with some great collaborators – Gene Tsudik, Ersin Uzun, Tao Wang, Ian Goldberg, Oleksii Starov, Bridger Hahn, Sheharbano Khattak, Julia Powles, and Emiliano De Cristofaro. I also have to thank Rami for making me see the light by introducing me to Python and Jupyter.

I want to thank the “Thirsty Thursdays” crew at Stony Brook – Sam, Shikha, Bryan, Connor, Ortega, and the many others. I hope the tradition carries on. I especially want to thank Sam and Shikha for their help during all my moves over the last couple of years. Being part of their American and Indian weddings was one of the highlights of my grad-school years.

I’ve been incredibly fortunate to have lived and traveled with many fun people over the last few years. I’d especially like to thank the folks from the 155 MTR days (Danielle and Dev), the Holly Berry house (Bill, Billy, Hau, and Sam), the Barnum house (Ainara, Cagla, Julia, Inigo, and Michael), and back home (Sandeep, Kavya, Monica, Sushmita, and Kiren).

I’d also like to thank the National Science Foundation and the Open Technology Fund for funding the research presented in this dissertation. Finally, I’d like to thank you, the reader, for being a contrarian and actually finding value in reading a dissertation. Of course, this applies only if you read past the acknowledgements.


As the Internet continues to integrate itself into the critical infrastructure and its users become increasingly dependent on it for commerce, communication, social and political organization, and many other services, it has become the primary source of information for private corporations and government agencies wanting to learn more (than is publicly available) about specific individuals and groups. This has led to an erosion of user privacy. For example, Internet Service Providers (ISPs) in the United States are able to monitor and sell data relating to customer browsing histories [1]. Additionally, the constantly lowering entry barrier to perform mass surveillance means that more government agencies have ramped up their efforts to monitor and silence their citizens – e.g., the Golden Shield Project in China [2], PRISM in the United States [3], and CMS/NETRA in India [4]. As a consequence of the absence of transparency and oversight in government surveillance programs and the scarcity of privacy-friendly corporations, many netizens have resorted to using privacy-enhancing tools such as the Tor client to protect themselves.
1.1 An Overview of Tor

Typically, when Internet users communicate with a server, the connections are established directly between the source (the Internet user) and the destination (the server). This enables routers and networks along the path to identify the source and destination of the flow, even if the communication is encrypted. The Tor client is a piece of software that aims to provide anonymity via unlinkability. It enables its users to communicate with Internet resources via the Tor network. The Tor network is a low-latency onion routing network with over 2M daily users and over 9K supporting servers, or relays. When a user connects to a destination server via the Tor client, the client establishes the connection using a nested and encrypted three relay circuit. The first relay, called the entry-relay, communicates directly with the Tor user. The last relay, called the exit-relay, communicates directly with the destination server. This circuit is constructed by the client one relay at a time, with a unique encryption key for each relay. The central idea is that no single observer is simultaneously aware of the identities – i.e., the IP address – of both, the source (user) and destination (server) of the connection. Since the exit relay communicates directly with the destination server, it is seen as the source of all (including, malicious) traffic exiting the Tor network. This may lead to its blocking or blacklisting by content providers. Therefore, relay operators have the capability to opt-out of serving as exit relays.

Each relay in the network publishes a server descriptor, which lists its IP address, Onion Routing port, bandwidth limitations (average and burst), and exit policy, among other things. The relay operator can adjust these variables. To ensure that no relay is overloaded, bandwidth limitations play a role in determining the probability of a relay being chosen as part of a circuit. Tor ensures that no relay is overloaded with more traffic than it is capable of handling. The exit policy allows operators who have chosen to run exit relays to limit the set of IP addresses and ports their relay will support. The server-descriptors published by relays are aggregated by directory authorities. The Tor directory authorities identify which positions in the circuit each relay may serve (e.g., only stable and fast relays may serve as entry relays and only relays configured to act as exits may serve as exit relays) and publish the aggregated list once per hour as the Tor consensus. The consensus is also available publicly through the CollecTor project.

Growth of the Tor network. Since 2010, the Tor network has experienced rapid growth in popularity amongst users, going from servicing 500K daily users with 4K volunteered relays in 890 different networks (Autonomous Systems) to 2M daily users and 9K relays in 1494 different networks. This rapid increase in size and popularity can be attributed to the increasing demand for anonymity and the clients ability to circumvent censorship. Figure 1.1 shows the increase in the capacity and size of the network since 2010. A side-effect of Tor becoming the de-facto tool for preserving anonymity and circumventing censorship is that it has also become a prime target for attacks.

---
1 We measure the number of active relays by counting the number of unique relays seen in a 24 hour time period.
1.2 Threats Faced by the Tor Network from Relays, Networks, and Users

The Tor anonymity network relies heavily on volunteer owned and operated resources to service millions of users each day. Consequently, it needs to manage these resources efficiently while addressing challenges to its robustness and utility. Many challenges faced by Tor arise from a deficit of trust in three entities – relay operators responsible for managing the relays through which Tor traffic flows, Autonomous Systems (ASes) that own the networks in which relays operate, and users interacting with the Internet via the Tor network. In general, the consequences of threats to the Tor network can be broadly classified into two categories: (1) compromised user anonymity by traffic correlation and (2) negative impact on network availability. Table 1.1 categorizes the threats faced by the Tor network and the consequences of each threat.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Threat</th>
<th>Demonstrated attacks</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relays</td>
<td>Website fingerprinting</td>
<td>[8, 9, 10, 11, 12, 13, 14]</td>
<td>Deanonymization</td>
</tr>
<tr>
<td></td>
<td>End-to-end confirmation</td>
<td>[15, 16, 17]</td>
<td>Deanonymization</td>
</tr>
<tr>
<td></td>
<td>Selective denial-of-service</td>
<td>[18, 19]</td>
<td>Deanonymization</td>
</tr>
<tr>
<td></td>
<td>Low-resource routing</td>
<td>[20]</td>
<td>Deanonymization</td>
</tr>
<tr>
<td>Networks</td>
<td>Traffic correlation</td>
<td>[21, 22, 23]</td>
<td>Deanonymization</td>
</tr>
<tr>
<td></td>
<td>Blocking access</td>
<td>[24, 25, 26, 27]</td>
<td>Reduced availability</td>
</tr>
<tr>
<td>Users</td>
<td>Abuse of anonymity</td>
<td>[28, 29]</td>
<td>Reduced availability</td>
</tr>
<tr>
<td></td>
<td>Resource exhaustion</td>
<td>[30]</td>
<td>Reduced availability</td>
</tr>
</tbody>
</table>

Table 1.1: The threats posed by each entity in the network and the consequences of each threat. Threats addressed in this dissertation, by way of measurements and/or defenses, are highlighted in bold.
1.2.1 Threats posed by malicious relays

At a high-level, Tor relays are responsible for forwarding user generated traffic from the source of the traffic, through the anonymity network, and to the destination. Since these relays are typically volunteer owned and operated, they may be malicious and perform active or passive attacks to deanonymize users. Malicious relays in the Tor network have several avenues for deanonymizing traffic passing through them.

**Website fingerprinting attacks.** Specific to Tor, these attacks allow the entry-relays in a circuit to identify websites being accessed by their users. In general, website fingerprinting attacks can be implemented by any adversary that has knowledge of the identity of the traffic source and is able to view traffic flow characteristics. These attacks work by correlating the flow characteristics of streams (e.g., packet sizes, directions, and inter-packet times) passing through them with websites known to generate similar flow characteristics. The general idea behind such attacks is: Flow characteristics generated by web-page loads (even with Tor and other encrypted transports) are unique and fingerprintable. Therefore, it is possible for an adversary with knowledge of the source of a Tor connection (e.g., entry-relays) to correlate observed flow characteristics with flows generated by monitored web-pages. Website fingerprinting attacks have been demonstrated to have reasonably high success rates in prior work [8, 9, 10, 11, 12, 13, 14]. Our work focuses on building defenses against these attacks.

**End-to-end confirmation techniques.** When a relay operator controls the entry-relay and exit-relay in a Tor circuit, deanonymization is possible — i.e., the entry-relay knows the source and the exit-relay knows the destination. The challenge is to identify when such a deanonymization opportunity is available. Since the exit-relay of a circuit is unaware of the corresponding entry-relay, a signaling mechanism is required to confirm that both relays belong to the same (or, colluding) operator(s). End-to-end confirmation techniques serve this purpose. Previous work [15, 16, 17] has relied on flow modification and error injection as a signaling mechanism between entry- and exit-relays. For example, a malicious entry-relay can modify Tor TCP streams to inject errors that can be detected by exit-relays in the circuit. Malicious exit-relays interpret these injected errors as a signal that the corresponding entry-relay is known and deanonymization is possible.

**Selective denial-of-service.** Selective denial-of-service attacks present a way for relay-level adversaries to maximize their resource utility. In this attack, malicious relays deny service to all clients utilizing circuits that it cannot deanonymize — e.g., they deny service to all circuits where the entry- and exit-relays are not owned by the same operator. Such attacks are used in conjunction with end-to-end confirmation signaling techniques. The attack was introduced in 2007 [18]. To combat selective denial-of-service attacks, the Tor network introduced the concept of guards [19]. However, since clients can be configured to not use guard relays, they can still be at risk to such attacks.

**Low-resource routing.** The Tor relay selection algorithm probabilistically selects relays in a circuit based on the bandwidth resources available to that relay. This allows the network to perform
load balancing and avoid performance issues stemming from relay overloading. As a consequence of this approach, the malicious relays can increase their likelihood of being selected in a circuit by falsely advertise high bandwidth capacities [20]. Recently, the Tor network introduced the concept of bandwidth authorities to regularly verify relay advertised capacities. This verification procedure reduces the window of opportunity for attackers, but does not eliminate it. Since the goal of such attacks is deanonymization, low-resource routing attacks need to be used in conjunction with end-to-end confirmation signaling techniques to identify when the goal is achievable.

In general, deanonymization attacks from relay-level adversaries are composed of an approach to increase likelihood of selection by the targeted Tor client, a signaling technique to identify when deanonymization is possible, and a correlation mechanism to complete deanonymization by exactly identifying the source and destination of the flow. Consider the case of website fingerprinting attacks. Here, a selective denial-of-service or low-resource routing attack may be used to increase the probability of the malicious relay being selected as an entry-relay, there is no need for a complex signaling technique since the controlling the entry-relay is sufficient for the attack to be launched, and the attack is complete when a high correlation between known website flow fingerprints and the observed flows is observed. In the case of more traditional deanonymization attacks, more complex signaling techniques are required, but the correlation mechanism is trivial since the exact identity of the source and destination is already known.

1.2.2 Threats posed by malicious networks

The main external threat to the Tor network comes from network-level adversaries such as Autonomous Systems (ASes), Internet Service Providers (ISPs), and Internet eXchange Points (IXPs). Network-level adversaries are capable of observing all traffic flowing in to, through, and out of their networks. They pose two primary challenges to the Tor network – correlation attacks to deanonymize flows and blocking to prevent use of the network.

**Traffic correlation attacks.** Network-level adversaries have the potential to perform deanonymization (at scale) via traffic correlation even without running Tor relays. To perform traffic correlation attacks, a network-level adversary only needs to be able to observe (and correlate) traffic entering the Tor network and exiting it. By proposing high accuracy flow correlation techniques, recent work [21, 22, 23] has shown that the threat posed by such attackers is higher than previously thought. Additionally, the proposed attacks also demonstrate how network-level adversaries can exploit Internet routing dynamics – *i.e.*, how network-level adversaries can exploit asymmetric routing behaviour and BGP insecurities to place themselves on paths to and from high-capacity relays in the Tor network. Such attacks increase the potential for large-scale (and targeted) deanonymization. Like relay-level adversaries, such attacks are also composed of approaches to increase likelihood of selection by the targeted Tor client (*e.g.*, exploiting BGP dynamics) and a correlation mechanism to complete deanonymization by exactly identifying the source and destination of the flow. The Tor client currently does not consider such attacks in their threat model [31] and therefore has no defense against them.
CHAPTER 1. INTRODUCTION

Blocking access to Tor. Recent work \[24, 25, 26, 27\] has highlighted and measured the blocking of Internet services (such as Tor) by network- and state-level adversaries. Tor relay IP addresses are publicly available and connections to them are easily identifiable. This allows network-level adversaries to identify and block Tor traffic originating in their networks (e.g., by the destination IP address of the connection request, cipher suites exchanged in the connection establishment phase, etc.). To prevent such blocking, the Tor project introduced pluggable transports \[7\] and unlisted entry relays called Bridges \[32\] in 2012. Pluggable transports and Tor bridges allow Tor connections to circumvent (or, raise the bar for) blocking by network-level adversaries by disguising their communication patterns \[33, 34, 35, 36\]. Network-level blocking adversaries and Tor developers are currently engaged in an arms race to block and develop new circumvention tools for Tor.

1.2.3 Threats posed by malicious users

Malicious Tor users can negatively impact the overall utility of network resources without investing in relay or network monitoring infrastructure. This can be done by reducing the amount of content accessible to exit-relays by abusing the anonymity provided by the Tor network or by resource exhaustion attacks on high-capacity relays.

Abuse of anonymity. The anonymity provided by the Tor network is a double-edged sword. While it allows users to maintain their privacy for legitimate purposes, it also allows malicious users to execute attacks on web servers under the guise of anonymity. Recent work by Khattak et al. \[28\] has shown that a large fraction of Tor relays face discrimination (e.g., through server-side blocking and CAPTCHAs) from content providers and network administrators. Reports from content providers \[37, 38\] justify these actions by citing the high volume of malicious traffic exiting the Tor network (as much as 94% of all Tor connections have been labeled as malicious by CloudFlare). This abuse of anonymity results in lower utility of the network – i.e., at its core, the problem is that in return for anonymity, each Tor user shares their reputation with other users. As a result, the malicious actions of a single Tor user can lead IP abuse blacklists to include IP addresses used by Tor exit relays. Consequently, websites and content providers treat even benign Tor users as malicious.

Resource exhaustion attacks. These attacks allow a malicious user to exhaust memory or computation resources available to a selected Tor relay, therefore preventing other Tor users from utilizing the relay in their circuits. One simple way to exhaust resources is to use botnets to create a large number of circuits through a single relay for spurious TCP connections. A more nuanced and low-resource attack is the Sniper attack by Jansen et al. \[30\]. The attack works by exploiting the Tor flow control mechanism as follows: The attacking client makes a request to a web server for a large amount of data using a circuit containing the victim relay as the entry to the network. The user then stops reading from the entry-relay TCP connection, causing the relay to buffer a limited amount of data. Then, to exhaust the entry-relay resources, the client tricks the exit-relay into sending more data through the circuit where it is again buffered by the entry-relay, eventually causing the relay to shut down due to insufficient memory.
1.3 Research Contributions

In this dissertation, we present our work to reduce the threat faced by Tor from three entities – relays that forward traffic through the Tor network, networks that provide the routing infrastructure used for communication on the Internet, and users who generate traffic that propagates through the Tor network. In particular, we present (1) traffic flow modification strategies to counter the threat from relay-level and eavesdropping adversaries, (2) relay selection strategies that utilize the state-of-the-art in Internet measurement to mitigate the threat of deanonymization by network-level adversaries, (3) an extensible covert-channel construction framework that addresses the threat of blocking by network-level adversaries by reversing the resource imbalance in the arms race between censors and circumvention tool developers, and (4) measurements that quantify server-side discrimination faced by legitimate Tor users as a consequence of abusive behavior from malicious users of the network. Table 1.2 summarizes our specific contributions in each area.

### Table 1.2: Summary of contributions and organization of this dissertation.

<table>
<thead>
<tr>
<th>Threat</th>
<th>Specific Contributions</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Website fingerprinting (§2)</td>
<td>CS-BuFLO defense</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>Theoretical foundations</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>Glove defense</td>
<td>2.5</td>
</tr>
<tr>
<td>Network-level traffic correlation (§3)</td>
<td>Measurement</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td>Cipollino defense</td>
<td>3.4</td>
</tr>
<tr>
<td>Network-level blocking (§4)</td>
<td>Castle defense</td>
<td>4.3</td>
</tr>
<tr>
<td>User abuse of anonymity (§5)</td>
<td>Synthetic measurement</td>
<td>5.4</td>
</tr>
<tr>
<td></td>
<td>User-driven measurement</td>
<td>5.5</td>
</tr>
</tbody>
</table>

1.3.1 Website fingerprinting defenses

We make three contributions. First, we propose a practical website fingerprinting defense – CS-BuFLO – that improves upon previous work by Dyer et al. [9]. Second, we build a theoretical framework for bootstrapping provably secure website fingerprinting defenses and evaluating existing defenses. Third, we leverage the constructed theoretical framework to build a provably secure and efficient fingerprinting defense – Glove.

**The Congestion-Sensitive BuFLO website fingerprinting defense.** We propose the CS-BuFLO (Congestion-Sensitive Buffered Fixed Length Obfuscator) website fingerprinting defense. The CS-BuFLO defense improves the real-world performance and security of a previously proposed defense, BuFLO, proposed by Dyer et al. [9]. At a high-level, CS-BuFLO presents a new approach in the design of fingerprinting defenses. Most previously-proposed defenses were designed in response to known attacks, and therefore take a black-listing approach to information leaks – i.e., they try to hide specific features, such as packet sizes from attackers. CS-BuFLO operates
by taking a white-listing approach. We start with a design that hides all traffic flow features and iteratively refine the design to reveal selected traffic flow features. This allows CS-BuFLO to make performance improvements without harming security against currently known fingerprinting attacks. The approaches proposed by CS-BuFLO, for improving practicality of website fingerprinting defenses, are currently being integrated into the Obfs proxy [39].

**Theoretical foundations of website fingerprinting defenses.** We develop theoretical foundations for modeling website fingerprinting attacks. With this framework, we focus on understanding the relationship between bandwidth overhead and security guarantees. We show how lower-bounds of the security vs. overhead trade-off can be obtained. We evaluate CS-BuFLO in the context of these results and show how comparative evaluations between any set of website fingerprinting defenses can be performed using the proposed framework. Additionally, we show how this framework allows us to make claims about the open-world performance of attacks and defenses given their closed-world evaluations.

**The provably secure Glove website fingerprinting defense.** Based on the results of our comparative and theoretical study of website fingerprinting defenses, we design a provably secure fingerprinting defense Glove. Glove demonstrates that efficient and secure website fingerprinting defenses are possible. The main idea behind Glove is simple: although webpages vary widely in size and structure, given a large set of webpages to be protected from fingerprinting, they can be clustered into groups of similar webpages. As a result, a website fingerprinting defense only needs to add a small amount of cover traffic to make all the pages in a cluster indistinguishable to an attacker. When a user loads a web page using such a defense scheme, the attacker can identify the cluster to which the page belongs, but gains no additional information about which page within that cluster the user is loading.

1.3.2 Network-level traffic correlation defenses

We make contributions in two dimensions. First, through a series of real-world and simulated experiments, we quantify the threat posed by traffic correlating network-level adversaries on the Tor network. Second, we develop a relay selection method to defend against such attacks by network-level adversaries.

**Quantifying the threat of traffic correlation by network-level adversaries.** We leverage the state-of-the-art in network measurement tools – PathCache [40]– to perform AS-level path prediction based on real-world control- and data-plane measurements. We fall back to algorithmic simulations [41] only when these measurements are unavailable. We find that our approach is more accurate than previously used path-prediction techniques. Next, we perform a current and a historical analysis to understand how the threat from AS-level adversaries has evolved over the past five years.

Our study on the current threat faced by Tor shows that around 30% of the circuits constructed by the Tor client are vulnerable to AS-level traffic correlation. Additionally, we find that Tor relays are currently vulnerable to BGP insecurities and two known to be malicious ASes in the network
have a BGP hijack and interception success rate of around 50% when attempting to hijack or intercept paths to and from Tor relays. We also find through these experiments that there are safe entry- and exit-relay combinations available during circuit construction, but these are not selected by the vanilla Tor client. This finding indicates that an AS-aware Tor client will be able to avoid AS-level traffic correlation in almost all cases.

From our historical analysis, we find that the threat faced by Tor clients has grown. Our experiments show that clients constructed between 3-14% more vulnerable circuits in 2015 than in 2010. These results are surprising given the massive growth in the size of the Tor network since 2010. This points to a fundamental problem with the Tor network – the lack of growth of AS-level diversity. We conclude that without specific efforts from the Tor project to increase diversity of relays or incorporate AS-awareness in the Tor client, our study shows that the threat is bound to increase.

**Defending against traffic correlation by network-level adversaries.** Motivated by the observation that vanilla Tor very often selects entry-exit pairs that may be subject to AS-level correlation attacks (even when there are an abundance of safe alternatives), we design an AS-aware Tor client – Cipollino– to mitigate the opportunities for such attackers.

We show that Cipollino is resilient to all currently known forms of traffic correlation attacks, including active attacks via BGP hijacks and interceptions. To accomplish this, Cipollino incorporates real-time data about control-plane anomalies so that it may avoid paths subject to on-going BGP hijacks or interceptions.

Our evaluation of Cipollino’s performance shows that the client provides a practical defense against traffic correlation attacks. Specifically, Cipollino (1) drastically improves the performance of AS-aware relay selection through a combination of pre-emptive circuit construction and aggressive circuit re-use, and (2) performs load-balancing to ensure that no set of Tor relays are over-loaded beyond their bandwidth capacities. We find that these performance enhancements also have the desirable side-effect of mitigating the threat faced from relay-level adversaries.

### 1.3.3 Circumventing network-level blocking

Our main contribution is an extensible covert-channel construction framework that addresses the threat of blocking by network-level adversaries by reversing the resource imbalance in the arms race between censors and circumvention tool developers. We implement this framework over video games. Two features make video games attractive for use as a cover protocol in censorship circumvention tools: First, games within a genre share many common features. Second, there are many different games, each with their own protocols and server infrastructures. These features allow circumvention tool developers to build a single framework that can be adapted to work with many different games within a genre; therefore allowing quick response to censor created blockades. In addition, censored users can diversify their covert communications across many different games, making it difficult for a censor to respond by simply blocking a single covert channel.

We demonstrate the feasibility of this approach by implementing our circumvention scheme
over three real-time strategy games (including two best-selling closed-source games). We evaluate
the security of our system prototype, Castle, by quantifying its resilience to a censor-adversary,
similarity to real game traffic, and ability to avoid common pitfalls in covert channel design. We
use our prototype to demonstrate that our approach can provide the throughput necessary for
bootstrapping higher bandwidth channels and also the transfer of textual data, such as web articles,
e-mail, SMS messages, and tweets, which are commonly used to organize political actions.

1.3.4 Measuring the impact of users abusing anonymity

We grapple with two key challenges: First, measuring Tor traffic is antithetical to the goals of
the anonymity system and presents ethical challenges. Second, defining and identifying undesired
or abusive network traffic is challenging as opinions vary and encryption can render inspection of
traffic infeasible. We address these challenges by focusing on the servers reactions to Tor traffic,
rather than the traffic itself – i.e., we take measurements of how servers react to Tor traffic using
synthetic and user-driven traffic.

To measure the prevalence of discrimination faced by users, we exercise multiple aspects of web-
sites and inspect them for subtle forms of discrimination (e.g., CAPTCHAs and interaction based
discrimination) in addition to outright blocking. To accurately measure discrimination against
users, we go beyond the prior work of Khattak et al. [28] and develop a crawler capable of exer-
cising the search and login features of websites. Taking measurements of real Tor traffic required
the creation and deployment of a privacy-sensitive logging approach for our own Tor exit relays.
We also consider aspects of Tor exit relays that make them more susceptible to complaints, IP
blacklisting, or blocking. We augment this analysis by deploying several Tor exits with varied
configurations and monitoring the reactions they produce.
In this chapter, we focus on defending Tor users against a class of traffic correlation attacks called website fingerprinting attacks. These attacks may be launched by Tor entry-relays and eavesdroppers that lie between a Tor client and the entry-relay. The results presented in this chapter have previously appeared as part of the following conference articles: [11, 12, 42, 43].
CHAPTER 2. WEBSITE FINGERPRINTING DEFENSES

2.1 Introduction

In a website fingerprinting attack, an adversary is able to monitor the communications between a client machine and a web browsing proxy. The traffic between the user and proxy may be encrypted, allowing the attacker to only observe the timing, direction, and size of packets exchanged between the client and proxy. Based on this information, the attacker attempts to infer the website(s) that the user is visiting via the proxy. The attacker may prepare for the attack by collecting information about specific websites in advance. For example, the attacker may visit websites using the same privacy service as the victim and collect a set of website fingerprints, which can then be used to identify the website(s) visited by the targeted client.

Website fingerprinting attacks have emerged as a serious threat against web browsing privacy mechanisms, such as Tor, SSL, and encrypting tunnels. Although these privacy mechanisms encrypt the content transferred between the web server and user, they do not effectively hide the users traffic flow characteristics, making them susceptible to such attacks by eavesdropping adversaries. Specifically, in the case of Tor, when a user browses the web, she reveals her traffic flow characteristics (e.g., packet sizes, inter-packet times, direction, etc.) to (1) intermediate routers between herself and the entry-relay and (2) relays selected to be a part of the client built circuit to her destination. This allows the intermediate routers and the entry-relay of the circuit to attempt to link the client IP address with websites known to generate similar traffic flows, thereby compromising client anonymity.

In this chapter, we present our work to defend Tor from such attacks. First, we present CS-BuFLO—a practical fingerprinting defense that is currently being integrated into the Obfs proxy [39]. Next, we develop a theoretical framework to model website fingerprinting attacks. We use this framework to obtain lower-bounds on the trade-off between the security provided by any website fingerprinting defense and its corresponding bandwidth and latency overhead. Finally, we use this framework to develop Glove—a provably secure website fingerprinting defense. Our proposed website fingerprinting defenses work by modifying traffic flow characteristics observable by the adversary. These defenses can be implemented by the entry-relays of the Tor network to prevent de-anonymization by eavesdropping adversaries between the client and entry-relays. To prevent such attacks by malicious entry-relays, the defense must also be enforced by the exit-relays.

2.2 Evolution of Website Fingerprinting Research

Traffic and timing analysis have repeatedly proven to be a useful technique for deducing information from encrypted network traffic [44, 45, 46, 47, 48, 49, 15, 50]. Our research focuses on a subset of such attacks known as website fingerprinting attacks. Researchers have engaged in a war of escalation in developing website fingerprinting attacks and defenses. In this section we outline the development of the research area.

\footnote{Other relays in the circuit are unable to identify the IP address of the Tor client without collusion with the entry-relay.}
Website fingerprinting attacks were first proposed on encrypted SSL and VPN tunnels \[51, 52, 53, 54\]. Most of these proposed attacks relied only on packet sizes to deduce information about the website being loaded. While they achieved some success in identifying webpages loaded by targeted clients when SSL was used, their success was limited against Tor traffic. This was because Tor pads all data packets to a multiple of 512 bytes, making packet sizes a less distinctive feature. As an example, Herrmann \textit{et al.} \[55\] proposed a website fingerprinting attack relying only on packet sizes and a Multinomial Naive-Bayes classifier. While their attack was able to achieve an accuracy of 94% against SSL encrypted tunnels, it obtained an accuracy of only 3% against Tor generated traffic flows. By incorporating information about packet ordering in their classifier, Shi \textit{et al.} were able to obtain an accuracy of 50% \[56\]. Similarly, Panchenko \textit{et al.} \[8\] were able to obtain an accuracy of 54% by using SVMs and feature vectors that included packet ordering and size features. To defend against these early attacks, network- and application-level defenses were proposed. Network-level defenses primarily involve modifying the traffic flow by padding, inserting, or splitting packets as they enter the network. Wright \textit{et al.} \[57\] proposed traffic morphing – a scheme where packet sizes are modified to match a given target distribution. The approach was extended by Lu \textit{et al.} \[54\] to matching n-gram packet distributions. Luo \textit{et al.} \[58\] proposed an application-level defense called HTTPOS. HTTPOS consisted of a collection of HTTP and TCP-level tricks for defending against website fingerprinting. Specifically, HTTPOS split single HTTP requests into multiple (possibly overlapping) requests while re-ordering requests and generating unnecessary requests. Additionally, TCP window sizes and MSS values were modified to force re-ordering of packets. The Tor project adopted a similar approach in 2011 \[59\].

In 2012, Cai \textit{et al.} \[10\] proposed the DL-SVM fingerprinting attack and demonstrated that it could achieve higher than 80% success against all previously proposed network- and application-level defenses. The DL-SVM attack proposed a new technique for classifying network traces (consisting of packet sizes, directions, and times). By considering network traces as a string and using the Damerau-Levenstein edit-distance to compute the similarity between two network traces, the attack was able to capture information about the ordering of packets and the requests that generated them. In the same year, Dyer \textit{et al.} \[9\] evaluated the traffic morphing approach along with other packet padding defenses such as padding packets to the network MTU, nearest power of two, randomly \textit{etc.}. They found that none of these approaches resulted in a significant drop in attacker accuracy when the attacker considered packet ordering as a classifier feature. In the same paper, Dyer \textit{et al.} proposed BuFLO (the Buffered Fixed Length Obfuscator). BuFLO considered modifications to packet sizes and ordering by padding (or, fragmenting) packets to a fixed size and injecting packets into the network at regular intervals. The BuFLO defense reduced all previous attack success rates to under 5% (the DL-SVM attack was not evaluated). However, this was at the cost of high bandwidth overhead (400%) due to injecting large quantities of padded \textit{junk} into the network. The DL-SVM attack was further improved by Wang and Goldberg in 2013, enabling a 95% attack success rate against Tor generated network traces \[60\]. Cai \textit{et al.} \[43\] also proposed CS-BuFLO, a defense based on the original BuFLO defense by Dyer \textit{et al.} \[9\]. The modifications were an effort
to make BuFLO more practical and require less overhead. The modifications proposed by the CS-BuFLO defense are currently being implemented as part of the Obfs proxy \[39\]. In 2014, Wang et al. \[12\] proposed the first open-world attack using a k-Nearest Neighbor classifier. In addition to considering an open-world evaluation, the evaluation also considered the impact of false-positives. The attack achieved a TPR of 0.85 and an FPR of .006. Following this, Nithyanand et al. \[42\] and Cai et al. \[11\] proposed the theoretical foundations of website fingerprinting and presented the first information-theoretically secure defenses against website fingerprinting attacks.

Since 2014, much of website fingerprinting research has been focused on making fingerprinting attacks and defenses more practical for the real-world. To this end, Wang and Goldberg \[14\] proposed a modification to existing provable defenses to lower bandwidth overheads. The major design modification enforced half-duplex communication in browsers – i.e., the browser can only send one request at a time, therefore reducing the number of features available to fingerprinting attackers. Wang and Goldberg \[13\] also proposed a new attack that was able to perform the task of parsing a single network trace into multiple network traces without compromising attack accuracy. Gu et al. \[61\] also demonstrated website fingerprinting attacks that were able to overcome the challenge of pages being loaded by a client in the multi-tab browsing setting. Finally, Kwon et al. \[62\] also demonstrated that website fingerprinting attacks are also able to identify hidden-services that are accessed by Tor clients. In particular, they were able to identify 50 different hidden-service servers with a TPR of .88 and FPR of .07 in the open-world setting.

2.3 The Congestion-Sensitive BuFLO Defense

In 2012, Dyer et al. \[9\] described BuFLO, a website fingerprinting defense scheme that hides all information about a website, except possibly its size, and performed a simulation-based evaluation that found that, although BuFLO is able to offer good security, it incurs a high cost to do so. In this section, we describe Congestion-Sensitive BuFLO (CS-BuFLO), an extension to BuFLO that includes numerous security and efficiency improvements. CS-BuFLO represents a new approach to the design of fingerprinting defenses. Most previously-proposed defenses were designed in response to known attacks, and therefore took a black-listing approach to information leaks, i.e. they tried to hide specific features, such as packet sizes. In designing CS-BuFLO, we take a white-listing approach we start with a design that hides all traffic features, and iteratively refine the design to reveal certain traffic features that enable us to achieve significant performance improvements without harming security.

2.3.1 An overview of the BuFLO defense

At a high level, the Buffered Fixed-Length Obfuscator (BuFLO) defense proposed by Dyer et al. \[9\] can be described as follows: BuFLO transmits a packet of size \(d\) bytes every \(\rho\) milliseconds, and continues doing so for at least \(\tau\) milliseconds. When less than \(d\) bytes of application data is available, then the packet is padded with junk data to ensure a size of exactly \(d\) bytes. The protocol
2.3. THE CONGESTION-SENSITIVE BUFLO DEFENSE

assumes that the junk bytes are marked so that the receiver can discard them. Similarly, if the website does not finish loading within $\tau$ milliseconds, then BuFLO continues transmitting until the website finishes loading and then stops immediately after. As described, BuFLO effectively hides everything about the website, except possibly its size. In spite of this, it has shortcomings that impact its practicality and security.

**Non-uniform security.** BuFLO either completely hides the size of the website or completely reveals it ($\pm d$ bytes). Thus it does not provide the same level of security to all websites.

**Uneven distribution of overhead.** BuFLO has large overheads for small websites. Thus its overhead is also unevenly distributed.

**TCP unfriendly and non-adaptive.** BuFLO is not TCP-friendly. In fact, it is the epitome of a bad network citizen. Further, BuFLO does not adapt when the user is visiting fast or slow websites. It wastes bandwidth when loading slow sites, and causes large latency when loading fast websites. BuFLO must be tuned to each user’s network connection. If the BuFLO bandwidth, \( \frac{1000d}{\rho} \) B/s, exceeds the user’s connection speed, then BuFLO will incur additional delay without improving security.

**Vulnerable to side-channels.** Past research by Fu et al., showed that transmitting at fixed intervals can reveal load information at the sender, which an attacker can use to infer partial information about the data being transmitted [63].

**Incompletely specified.** BuFLO is incompletely specified. In their description, Dyer et al. did not describe: (1) How BuFLO detects that a page load is complete and (2) how BuFLO handles bidirectional communication.

**Impractical.** As noted by its authors, BuFLO’s simulation based results “reflect an ideal implementation that assumes the feasibility of implementing fixed packet timing intervals. This is at the very least difficult and impossible for certain values of $\rho$. These simulations also ignore the complexities of cross-layer communication in the network stack. As a result, it remains unclear how well the defense performs in the real world.

Dyer et al. proposed BuFLO as a straw-man defense system, so it is understandable that they did not bother addressing these problems. However, we find that these problems have common solutions, e.g., we can simultaneously improve overhead and TCP-friendliness, simultaneously make security and overhead more uniform across a range of websites, etc. We implement these solutions in our proposed defense – CS-BuFLO.

2.3.2 The CS-BuFLO defense

Similar to BuFLO, CS-BuFLO delivers fixed-size chunks of data. However, CS-BuFLO randomizes the timing of network writes in order to counter the attack of Fu et al. [63]. In spite of performing randomized network writes, it maintains a target average inter-packet time, $\rho^*$. This parameter, $\rho^*$, is updated to match its bandwidth to the rate of the sender (see Section 2.3.2.1). Since updating $\rho^*$ based on the sender’s rate reveals information about the sender, CS-BuFLO performs these updates
infrequently. Additionally, CS-BuFLO uses TCP to be congestion friendly, and uses feedback from the TCP stack in order to reduce the amount of junk data it needs to send (see Section 2.3.2.2). Also like BuFLO, CS-BuFLO transmits extra junk data after the website has finished loading in order to hide the total size of the website (see Section 2.3.2.3). However, CS-BuFLO uses a scale-independent padding scheme and monitors the state of the page loading process to avoid unnecessary overheads (see Section 2.3.2.4). In Algorithm 2.1, we show the pseudo-code of the CS-BuFLO server implementation.

### 2.3.2.1 Dynamic rate adaptation

CS-BuFLO adapts its transmission rate to match the rate of the sender. This reduces wasted bandwidth when dealing with slow senders, and it reduces latency when dealing with fast senders. However, adapting CS-BuFLO’s transmission rate to match the sender’s reveals information about the sender, and therefore may harm security. To deal with this, CS-BuFLO takes certain precautions to limit the information leaked through rate adaptation. These are illustrated in Figure 2.1.

First, it only adapts after transmitting $2^k$ bytes, for some integer $k$. Thus, during a session in which CS-BuFLO transmits $n$ bytes, it will only perform $\log_2 n$ rate adjustments, limiting the information leakage. This choice also allows CS-BuFLO to adapt more quickly during the beginning of a session, when the sender is likely to be performing a TCP slow start. During this phase, CS-BuFLO is able to ramp up its transmission rate just as quickly as the sender can.

Further information leakage is avoided by using a robust statistic to update $\rho^*$. Between adjustments, CS-BuFLO collects estimates of the senders instantaneous bandwidth. It then sets $\rho^*$
so as to match the senders median instantaneous bandwidth. Median is a robust statistic, meaning that the new $\rho^*$ value will not be strongly influenced by bandwidth bursts and lulls, and hence $\rho^*$ will not reveal much about the sender’s transmission pattern. Note that the estimator only collects measurements during uninterrupted bursts from the sender. This ensures that the bandwidth measurements do not include delays caused by dependencies between requests and responses. For example, if the estimator sees a packet $p_1$ from the website, then a packet $p_2$ from the client, and then another packet $p_3$ from the website, it may be the case that $p_3$ is a response to $p_2$. In this case, the time between $p_1$ and $p_3$ is constrained by the round trip time, not the website’s bandwidth. Finally, CS-BuFLO rounds all $\rho^*$ values up to a power of two. This further hides information about the sender’s true rate, and gives the sender room to increase its transmission rate, e.g., during slow start. The complete algorithm for estimating the $\rho^*$ parameter used in CS-BuFLO’s rate adaptation mechanism is specified in Algorithm 2.2.

### 2.3.2.2 Congestion-sensitivity

We use a simple idea to make CS-BuFLO congestion sensitive and TCP friendly: We run the protocol over TCP! In addition, we increase efficiency of the defense by adjusting the amount of junk data padding in accordance to network congestion.

Algorithm 2.3 shows our method for taking advantage of congestion to reduce the amount of junk data sent by CS-BuFLO. Note first that cs-send always writes exactly $d$ bytes to the TCP socket. Since the amount of data presented to the TCP socket is always the same, this algorithm reveals no information about the timing or size of application-data packets from the website that have arrived at the CS-BuFLO proxy. To understand how the algorithm works, imagine the TCP connection to the client stalls for an extended period of time. Eventually, the kernel’s TCP send queue for socket $s$ will fill up, and the call to write will return 0. From then until the TCP congestion clears up, all CS-BuFLO calls to cs-send will not append any further junk data to $B$.

### 2.3.2.3 Stream padding

CS-BuFLO hides the total size of real data transmitted by continuing to transmit extra junk data after the browser and web server have stopped transmitting.

Table 2.1 shows two padding schemes we experimented with in CS-BuFLO. Both schemes introduce at most a constant factor of additional cost, but reveal at most a logarithmic amount of information about the size of the website. The first scheme, which we call payload padding, continues transmitting until the total amount of transmitted data ($R + J$) is a multiple of $2^{\left\lfloor \log_2 R \right\rfloor}$. This padding scheme will transmit at most $2^{\left\lfloor \log_2 R \right\rfloor}$ additional bytes, so it increases the cost by at most a factor of 2, but it reveals only $\log_2 R$. The second scheme, which we call total padding, continues transmitting until $R + J$ is a power of 2. This also increases the cost by at most a factor of 2 and reveals, in the worst case, $\log_2 R$. As we see in our empirical evaluation, this approach hides more information about $R$ than payload padding.
Table 2.1: CS-BuFLO: Padding mechanisms. The CS-BuFLO client and server can use either approach – payload or total padding, independently of the choice of the other end.

Note that the CS-BuFLO server and client do not have to use the same stream padding scheme. Thus, there are four possible padding configurations, which we denote as CPSP (client payload, server payload), CPST (client payload, server total), CTSP (client total, server payload) and CTST (client total, server total).

In order to determine when to stop padding, the CS-BuFLO server must know when the website has finished transmitting. CS-BuFLO uses two mechanisms to recognize that the page has finished loading. First, the CS-BuFLO client proxy monitors for the browser’s onLoad event. The client notifies the CS-BuFLO server when it receives the onLoad event from the browser. Once the server receives the onLoad message from the client, it considers the web server to be idle (see Algorithm 2.4) and will stop transmitting as soon as it adds sufficient stream padding and empties its transmit buffer. As a backup mechanism, the server considers the website idle if quiet-time seconds pass without receiving new data from the website. We used a quiet-time of 2 seconds in our prototype implementation.

2.3.2.4 Termination and page-load detection

The CS-BuFLO server is likely to finish each page load by sending a long tail of junk packets. This tail can be a significant source of overhead and may not provide much additional security, as demonstrated by our evaluation.

Our initial investigations revealed that the long tail served two purposes which could also be served through other, more efficient means. First, the long tail helps hide the total size of the website. However, the interior padding performed by cs-send also obscures the total size of the website. Second, in the specific context of web browsing, the long tail also hides the size of the last object sent from the web server to the client. The attacker can infer some information about the size of this object by measuring the amount of data the CS-BuFLO server sends to the CS-BuFLO client after the client stops transmitting to the server. However, we can obscure this information by having the client continue to send junk packets to the server, i.e., more aggressive stream padding from the client may obviate the need for aggressive padding at the server while improving security. Based on these ideas, we implemented an early termination feature in our CS-BuFLO prototype. Here, the CS-BuFLO client notifies the server that it is done padding. After receiving this message, the CS-BuFLO server will stop transmitting as soon as the web server becomes idle and its buffers are empty.
Figure 2.2: CS-BuFLO: Early termination. More padding at the client can help hide the size of the last object sent from the server to the client. Early termination can avoid unnecessary padding at the end of a page load.

Figure 2.2 illustrates how the padding schemes used by the client and server interact while considering the impact of early termination. Additional client padding can hide the size of the last HTTP object, and early termination can avoid unnecessary padding.

### 2.3.2.5 Fixing packet sizes

Sending fixed-length packets hides packet size information from the attacker. Although any fixed length should work, it is important to choose a packet length that maximizes performance. Since we may transmit packets consisting entirely of junk data during the transmission, larger packets will result in higher bandwidth overhead, while smaller packets can result in longer loading times due to unsaturated links. To select a uniform packet size for CS-BuFLO, an experiment was conducted to understand the distribution of packets generated by web-browsing. These experiments revealed that over 95.7% of all upstream packet transmissions were under 600 bytes. We selected 600 bytes as the standard packet size for our CS-BuFLO implementation.

### 2.3.3 Prototype implementation

CS-BuFLO was implemented as a modification to OpenSSH-5.9p1. The SSH client was modified to accept a new SOCKS proxy command code, `onLoadCmd`, to indicate stream termination to the server. A Firefox plugin, `OnloadNotify`, was also developed. The plugin was used to indicate to the client that the webpage load was complete. The following OpenSSH message types were used in our implementation:

1. The OpenSSH message type `SSH_MSG_IGNORE`, which means all payload in a packet of this type can be ignored, was used to insert junk data whenever needed.
2. The `SSH_MSG_NOTIFY_ONLOAD` message was created to be used by the client to communicate reception of `onLoadCmd` from the browser, to the server. When this message is received by
the CS-BuFLO server, it stops transmitting as soon as it empties its buffer (and completes sufficient stream padding).

3. The `SSH_MSG_NOTIFY_PADDINGDONE` message was created to implement the early termination feature of CS-BuFLO. Upon receiving this message from the client, the CS-BuFLO server stops transmitting as soon as the web server becomes idle and its buffers are empty.

All the above messages were buffered and transmitted just like other messages in Algorithm 2.1, i.e., using `cs-send`, therefore an attacker is unable distinguish these messages from other traffic.

Algorithm 2.1 The CS-BuFLO server.

```plaintext
function CSBuFLO-Server(s)
  while true do
    (m, ρ) = READ-MESSAGE(ρ)
    if m is application data from website then
      output-buff ← output-buff ∥ data
      real-bytes ← real-bytes + LENGTH(m)
      last-site-response-time ← CURRENT-TIME
    else if m is application data from client then
      send m to the website
      ρ-stats ← ρ-stats ∥ ⊥
      onLoadEvent ← 0, padding-done ← 0
    else if m is onLoad message then
      onLoadEvent ← 1
    else if m is padding-done message then
      padding-done ← 1
    else if m is a time-out then
      output-buff-bytes ← LENGTH(output-buff)
      (output-buff, j) ← CS-SEND(s, output-buff)
      junk-bytes ← junk-bytes + j
      real-bytes-sent ←
      real-bytes-sent + output-buff-bytes - LENGTH(output-buff) - j
      if real-bytes-sent ≥ PACKET-SIZE then
        real-bytes-sent ← real-bytes-sent - PACKET-SIZE
        ρ-stats ← ρ-stats ∥ CURRENT-TIME
      end if
    end if
    if DONE-XMITTING then
      reset all variables
    else
      if ρ* = ∞ then
        ρ* ← INITIAL-RHO
      else if CROSSED-THRESHOLD(real-bytes + junk-bytes) then
        ρ* ← RHO-ESTIMATOR(ρ-stats, ρ*)
        ρ-stats ← ∅
      end if
      if m is a time-out then
        ρ ← random number in [0, 2ρ*]
      end if
    end if
  end while
end function
```
2.3. THE CONGESTION-SENSITIVE BUFLO DEFENSE

Algorithm 2.2 Estimating new value of $\rho^*$ based on network performance.

\begin{verbatim}
function RHO-ESTIMATOR($\rho$-stats, $\rho^*$)
    $I \leftarrow \{\rho$-stats$_i, \rho$-stats$_{i+1} \mid \rho$-stats$_i \neq \bot \land \rho$-stats$_{i+1} \neq \bot\}$
    if $I$ is empty list then
        return $\rho^*$
    else
        return $2^\lceil \log_2 \text{median}(I) \rceil$
    end if
end function
\end{verbatim}

Algorithm 2.3 Algorithm for sending data and using feedback from TCP using non-blocking sockets.

\begin{verbatim}
function CS-SEND(socket $s$, output-buff)
    $n \leftarrow \text{LENGTH}(output-buff)$
    $j \leftarrow 0$
    if $n < \text{PACKET-SIZE}$ then
        $j \leftarrow \text{PACKET-SIZE} - n$
        output-buff $\leftarrow$ output-buff $\parallel j$
    end if
    $r \leftarrow \text{write}(s, output-buff, \text{PACKET-SIZE})$
    if $r \geq n$ then
        $\triangleright$ Optional: reclaim unsent junk
        output-buff $\leftarrow$ empty buffer
        $j \leftarrow r - n$
    else
        remove last $j$ bytes from output-buff
        remove first $r$ bytes from output-buff
        $j \leftarrow 0$
    end if
    return (output-buff, $j$)
end function
\end{verbatim}

Algorithm 2.4 The CS-BuFLO termination algorithm.

\begin{verbatim}
function DONE-XMITTING
    return \text{LENGTH}(output-buff) $\leftarrow$
    0 $\land$ \text{CHANNEL-IDLE}(onLoadEvent, last-site-response-time) $\land$
    \text{CROSSED-THRESHOLD}(real-bytes + junk-bytes)
end function

function CHANNEL-IDLE(onLoadEvent, last-site-response-time)
    return onLoadEvent $\lor$
    (last-site-response-time + \text{QUIET-TIME} < \text{CURRENT-TIME})
end function

function CROSSED-THRESHOLD($x$)
    return $\lfloor \log_2(x - \text{PACKET-SIZE}) \rfloor < \lfloor \log_2 x \rfloor$
end function
\end{verbatim}
2.4 Theoretical Foundations

In this section we develop a model of website fingerprinting attacks and defenses, derive lower-bounds on the bandwidth overhead of any defense that achieves a given level of security, and show how to derive open-world performance from closed-world experimental results.

2.4.1 Security vs. Overhead Trade-Off

We focus on understanding the relationship between bandwidth overhead and security guarantees. The overhead required by a fingerprinting defense depends on the set of web sites to be protected – a set of similar websites can be protected with little overhead, a set of dissimilar websites requires more overhead. To derive lower-bounds, we consider an offline version of the website fingerprinting defense problem, i.e. the defense system knows, in advance, the set of websites that the user may visit and the packet traces that each website may generate. We develop an efficient dynamic program to compute a lower-bound on the bandwidth overhead of any fingerprinting defense scheme in the closed-world setting.

2.4.1.1 Definitions

In a website fingerprinting attack, the defender selects a website, $w$, and uses the defense mechanism to load the website, producing a packet trace, $t$, that is observed by the attacker. The attacker then attempts to guess $w$.

Let $W$ be a random variable representing the URL of the website selected by the defender. The probability distribution of $W$ reflects the probability that the defender visits each website. For each website, $w$, let $T^D_w$ and $T_w$ be the random variables representing the packet trace generated by loading $w$ with and without defense system $D$, respectively. Packet traces include the time, direction, and content of each packet. Since cryptographic attacks are out of scope for our work, we assume any encryption functions used by the defense scheme are information-theoretically secure. The probability distribution of $T^D_w$ captures variations in network conditions, changes in dynamically-generated web pages, randomness in the browser, and randomness in the defense system. We assume the attacker knows the distribution of $W$ and $T^D_w$ for every $w$.

In a closed world setting, the attacker’s goal is to infer $W$ from $T^D_w$. The optimal closed-world attacker, $A$, upon observing trace $t$, outputs

$$A(t) = \arg \max_w \Pr[W = w] \Pr[T^D_w = t]$$

If more than one $w$ attains the maximum, then the attacker chooses randomly among them.

Some privacy applications require good worst-case performance, and some only require good average-case performance. This leads to two security definitions for website fingerprinting defenses:

**Definition 1.** A fingerprinting defense $D$ is non-uniformly $\epsilon$-secure for $W$ iff $\Pr[A(T^D_w) = W] \leq \epsilon$. Defense $D$ is uniformly $\epsilon$-secure for $W$ if $\max_w \Pr[A(T^D_w) = w] \leq \epsilon$. 
These are information-theoretic security definitions – $A$ is the optimal attacker described above. The first definition says that $A$’s average success rate is less than $\epsilon$, but it does not require that every website be difficult to recognize. The second definition requires all websites to be at least $\epsilon$ difficult to recognize. All previous papers on website fingerprinting attacks and defenses have reported average attack success rates in the closed-world model, i.e. they have reported non-uniform security measurements. We will do the same.

To define the bandwidth overhead of a defense system, let $B(t)$ be the total number of bytes transmitted in trace $t$. We define the bandwidth ratio of defense $D$ as:

$$BWRatio_D(W) = \frac{E[B(T^D_W)]]}{E[B(T_W)]}$$

This definition captures the overall bandwidth ratio between a user surfing the web while using defense $D$ and a user visiting the same websites with no defense.

### 2.4.1.2 Bandwidth lower-bounds

We now derive an algorithm to compute, given websites $w_1, \ldots, w_n$, a lower-bound for the bandwidth that any non-uniformly $\epsilon$-secure fingerprinting defense can use in a closed-world experiment using $w_1, \ldots, w_n$. To compute this lower-bound on bandwidth, we consider an adversary that looks only at the total number of bytes in a packet trace, i.e., an attacker $A_S$ that always guesses

$$A_S(t) = \arg\max_w \Pr[B(T^D_w) = B(t)]$$

Any defense that is $\epsilon$-secure against an arbitrary attacker must also be at least $\epsilon$-secure against $A_S$. If we can derive a lower-bound on defenses that are $\epsilon$-secure against $A_S$, that lower-bound will apply to any $\epsilon$-secure defense.

We make two simplifying assumptions in order to obtain an efficient algorithm for computing lower-bounds. First, we assume that each website has a unique fixed size, $s_i$. In our closed-world experiments, we found that just over half the web pages in our dataset had a normalized standard deviation of less than 0.11 across 20 loads, so we do not believe this assumption will impact the results of our analysis. Second, we assume that the defense mechanism does not compress or truncate the website.

Suppose websites $w_1, \ldots, w_n$ have sizes $s_1 < s_2 < \ldots < s_n$. Let $S = \{s_1, \ldots, s_n\}$. For any defense, $D$, let $p_{ij}$ be the probability that $D$ transmits $j$ bytes during a load of website $w_i$. Since, in a closed-world experiment, each website occurs with probability $1/n$, the bandwidth cost of $D$ is:

$$\sum_{j=1}^{\infty} \sum_{i=1}^{n} \frac{1}{n} j p_{ij}$$
CHAPTER 2. WEBSITE FINGERPRINTING DEFENSES

and the non-uniform success probability of \( A_S \) is:

\[
\sum_{j=1}^{\infty} \frac{\max_i p_{ij}}{\sum_i p_{ij}} \cdot \frac{\sum_i p_{ij}}{n} = \sum_{j=1}^{\infty} \frac{\max_i p_{ij}}{n}
\]

We derive lower-bounds on the bandwidth cost of \( D \) by computing the matrix of \( p_{ij} \) values that minimize the above bandwidth cost function while still satisfying the above security constraint. Recall that, since \( D \) is assumed not to compress or truncate web pages, \( p_{ij} = 0 \) for \( j < s_i \). We propose two theorems before we present our algorithm for computing the bandwidth lower-bound for uniformly and non-uniformly \( \epsilon \)-secure fingerprinting defenses.

**Theorem 1.** Suppose \( \epsilon n \) is an integer. Let \( W \) be a random variable uniformly distributed over \( w_1, \ldots, w_n \), i.e. \( W \) represents a closed-world experiment. Suppose \( D \) is a defense that is \( \epsilon \)-non-uniformly-secure against \( A_S \) on distribution \( W \). Then there exists a monotonically increasing function \( f \) from \( S = \{ s_1, \ldots, s_n \} \) to itself such that

\[
\begin{align*}
|f(S)| &\leq \epsilon n, \\
\sum_{i=1}^{n} f(s_i)/\sum_{i=1}^{n} s_i &\leq \text{BWRatio}_D(W).
\end{align*}
\]

**Proof.** The overall structure of the proof for non-uniform security is: First we constrain the structure of the optimal \( p_{ij} \) so that we can formulate the optimization problem as a linear program. Next, we prove that the linear program has an integral solution, so that the optimal solution is equivalent to a function \( f : S \rightarrow S \) satisfying certain constraints. Finally, we show that \( f \) is monotonically increasing.

**Linear program formulation.** Let \( p_{ij} \) be the probabilities that minimize the bandwidth cost while meeting the security requirement. We then can make the following statements (proofs supplied below each statement):

1. \( p_{ij} = 0 \) unless \( j \in \{ s_1, \ldots, s_n \} \). Here we claim that the optimal defense does not map websites to sizes outside the set of sizes of all websites in the closed-world.

   Suppose \( p_{lj} \neq 0 \) where \( s_k < j < s_{k+1} \). Then we can make a more efficient and no less secure by replacing \( p_{is_k} \) with \( p_{is_k} + p_{ij} \) for all \( i \) and setting \( p_{ij} = 0 \) for all \( i \). This will have lower bandwidth cost because \( s_k < j \). This will not violate the constraint that, for all \( i \) and \( j' < s_i \), \( p_{ij'} = 0 \), because, if \( p_{ij} \neq 0 \) before the change, then \( s_i \leq j \), so \( s_i \leq s_k \). This will not worsen security because \( \max_i (p_{is_k} + p_{ij}) \leq \max_i p_{is_k} + \max_i p_{ij} \).

2. If \( p_{ij} < \max_k p_{kj} \), then for all \( j' > j \), \( p_{ij'} = 0 \). Here we claim that for the optimal defense, if a website is mapped to some size \( s \), then this mapping occurs with the same probability as any other non-zero mapping to size \( s \).

   Suppose otherwise. Let \( t = \min(\max_k p_{kj} - p_{ij}, p_{ij'}) \). Note \( t \neq 0 \). Thus we can construct a more efficient and no less secure defense by replacing \( p_{ij} \) with \( p_{ij} + t \) and \( p_{ij'} \) with \( p_{ij'} - t \).
3. For all \( j \), \( p_{kj} \leq p_{k+1,j} \) for \( k \in [1,i] \), where \( s_i \leq j < s_{i+1} \). Here we claim that it is not possible for a larger website to have a higher probability mapping to a given size than a smaller website with non-zero probability mapping to the same size.

Suppose \( p_{kj} > p_{k+1,j} \) for some \( k \in [1,i] \), where \( s_i \leq j < s_{i+1} \). This implies that \( p_{k+1,j} \leq \max_j p_{ij} \). By Item 2, \( p_{k+1,j'} = 0 \) for all \( j' > j \). Thus we must have that: \( \sum_{j'=s_{k+1}}^j p_{k+1,j'} = 1 \). This also implies that \( p_{kj} = 0 \). Thus, by Item 2, \( p_{kj'} = \max_j p_{ij} \) for all \( j' \in \{s_k, \ldots, j-1\} \).

This implies that \( \sum_{j'=s_k}^j p_{kj'} > \sum_{j'=s_{k+1}}^j p_{k+1,j'} = 1 \), a contradiction.

Since \( p_{ij} \) is non-zero only if \( j \in \{s_1, \ldots, s_n\} \), we can relabel the \( p_{ij} \) to be the probability that the defense transmits \( s_j \) bytes during a load of website \( w_i \). The above claims imply that \( \max_i p_{ij} = p_{ii} \), so the security constraint can be re-written as \( \sum_{i=1}^n p_{ii} \leq \epsilon n \).

Now that the security constraint is a linear function of the \( p_{ij} \) variables, we can formulate a linear program for computing the optimal \( p_{ij} \) values:

\[
\text{minimize } \sum_{i=1}^n \sum_{j=i}^n p_{ij}s_j \quad \text{(the bandwidth cost)}
\]

subject to the constraints

\[
(a) \quad \sum_{i=1}^n p_{ii} \leq \epsilon n \quad \text{(\( \epsilon \) non-uniform security)}
\]
\[
(b) \quad \sum_{j=i}^n p_{ij} = 1 \quad \text{\( (p_{ij} \) are probabilities)}
\]
\[
(c) \quad 0 \leq p_{ij} \leq 1
\]

**Proving existence of an integral solution.** Linear programs with Totally Unimodular (TU) constraint matrices and integral objective functions have integral solutions [64]. We prove that the constraint matrix, \( A \) (derived by the constraints (a), (b), and (c) of the above LP), is TU. To prove TU-ness of \( A \), it is sufficient to prove the following [63]: (i) Every column contains at-most 2 non-zero entries, (ii) Every entry is 0, 1, or -1, (iii) If two non-zero entries in any column of \( A \) have the same sign, then the row of each belongs in two disjoint partitions of \( A \).

Since the set of TU matrices is closed under the operation of adding a row or column with at-most one non-zero entry [69], we may delete the 2\( n \) rows of \( A \) corresponding to constraint (c) and prove that the remaining constraint matrix \( A' \) satisfies the TU conditions (i) - (iii).

Observe the following properties of \( A' \):

- There are \( n \) rows (WLOG, rows 1 to \( n \)) induced by the constraint (a). These are such that: \( A_{i,(i-1)n}, \ldots, A_{i,in-1} = 1, \forall i \in \{1, \ldots, n\} \) and 0 for all other entries. Therefore, each column of the partition \( B \) composed of these \( n \) rows contains only a single non-zero entry (i.e., +1).
- There is only 1 row (WLOG, row \( n+1 \)) induced by the constraint (b). This row has the form: \( A_{n+1,j} = 1, \forall j \in \{1^2, \ldots, n^2\} \) and 0 for all other entries. Each column of the partition \( C \) composed of this single vector may contain only a single non-zero entry (i.e., +1).
CHAPTER 2. WEBSITE FINGERPRINTING DEFENSES

From the above properties, it is clear that matrix $A'$ is TU since: Each column contains at-most two non-zero entries (+1) and it may be partitioned into matrices $B$ and $C$ such that condition (iii) is satisfied. Therefore, the matrices $A'$ and $A$ are TU and the LP describing $A$ has only integral optima. Therefore the linear program has an integral solution – i.e., the optimal fingerprinting defense is deterministic.

Proving existence of a monotonic function $f$. In an integral solution of the linear program, all the probabilities are 0 or 1, so the solution is equivalent to a function $f : S \rightarrow S$ satisfying:

- $|f(S)| \leq \epsilon n$.
- $\sum_{i=1}^{n} f(s_i) / \sum_{i=1}^{n} s_i \leq \text{BWRatio}_D(W)$.

We now show that this mapping function $f$ not only exists, but also corresponds to the optimal non-uniformly $\epsilon$-secure defense is monotonic.

Consider any partition of $\{s_1, \ldots, s_n\}$ into sets $S_1, \ldots, S_k$. Let $m_i = \max_{s \in S_i} s_i$. Without loss of generality, assume $m_1 \leq m_2 \leq \cdots \leq m_k$. Now consider the monotonic allocation of traces into sets $S^*_1, \ldots, S^*_k$ where $|S^*_i| = |S_i|$. Let $m^*_i = \max_{s \in S^*_i} s$. Observe that $m^*_i \leq m_i$ for all $i$, i.e., the new allocation has lower bandwidth. Since the number of sets in the partition and the sizes of those sets are unchanged, this new allocation has the same security as the original, but lower bandwidth. Hence the optimal $f$ must be monotonic.

Intuitively, $f$ represents a mapping from each website’s original size ($s_i$) to the number of bytes that $D$ transmits when loading website $w_i$. The above theorem shows that this function is monotonic and deterministic. In addition, it enables us to efficiently compute a lower-bound on the overhead of any defense that is $\epsilon$ non-uniformly secure in a closed world experiment on $w_1, \ldots, w_n$. To get a lower-bound for non-uniformly $\epsilon$-secure defenses, we just need to find a monotonically increasing function $f : S \rightarrow S$ that satisfies $|f(S)| \leq \epsilon n$ and minimizes $\sum_{i=1}^{n} f(s_i)$.

Such an $f$ is equivalent to a partition $S_1, \ldots, S_k$ of $S$ satisfying $k \leq \epsilon n$ and minimizing $\sum_{i=1}^{k} |S_i| \max_{s \in S_i} s$. These partitions satisfy a recurrence relation. If $S_1, \ldots, S_k$ is an optimal non-uniformly $\epsilon/n$-secure partition, then $S_1, \ldots, S_{k-1}$ is an optimal non-uniformly $\epsilon/n - |S_k|$-secure partition of $S_1 \cup \cdots \cup S_{k-1}$. Therefore the cost, $C(k n, n)$, of the optimal $f$ satisfies the recurrence:

$$C(k n, n) = \begin{cases} ns_n & \text{if } k = 1 \\ \min_{1 \leq j \leq n-1} C(k-1 n-j, n-j) + js_n & \text{otherwise.} \end{cases}$$

By a similar theorem, we can obtain a lower-bound for uniformly $\epsilon$-secure deterministic defenses.

**Theorem 2.** Let $W$ be uniformly distributed over $w_1, \ldots, w_n$, i.e. $W$ represents a closed-world experiment. Suppose $D$ is a deterministic defense that is uniformly $\epsilon$-secure against $A_S$ on distribution $W$. Then there exists a monotonically increasing function $f$ from $S = \{s_1, \ldots, s_n\}$ to itself such that:
2.4. THEORETICAL FOUNDATIONS

- \(\min_i |f^{-1}(s_i)| \geq 1/\epsilon.\)
- \(\sum_{i=1}^n f(s_i)/\sum_{i=1}^n s_i \leq \text{BWRatio}_D(W).\)

Proof. The proof is almost identical to Theorem 1. We only need to modify the final part which proves the existence of the function \(f\). We illustrate here that there is a similar function for any deterministic uniformly secure defense \(D\). Set \(f(s_i) = b_i\) where \(b_i\) is the number of bytes transmitted when the defense \(D\) loads website \(w_i\). Since \(D\) does not compress or truncate websites, we must have \(b_i \geq \max_{s \in f^{-1}(b_i)} s\) for all \(i\). Observe that we can assume \(b_i = \max_{s \in f^{-1}(b_i)} s\) without harming security or efficiency, so that \(f : S \rightarrow S\). Thus \(f\) satisfies the security constraint \(\min_i |f^{-1}(s_i)| \geq 1/\epsilon,\) and \(\sum_{i=1}^n f(s_i)/\sum_{i=1}^n s_i \leq \text{BWRatio}_D(W).\)

As with the lower-bound on non-uniformly secure defenses, such an \(f\) corresponds to a partition \(S_1, \ldots, S_k\) of \(S\) satisfying \(\min_i |S_i| \geq 1/\epsilon\) and minimizing \(\sum_{i=1}^k |S_i| \max_{s \in S_i} s\). These partitions satisfy a slightly different recurrence. If \(S_1, \ldots, S_k\) is an optimal uniformly \(\epsilon\)-secure partition of \(S\), then \(S_1, \ldots, S_{k-1}\) is an optimal uniformly \(\epsilon\)-secure partition on \(S_1 \cup \cdots \cup S_{k-1}\). Thus the cost, \(C(\epsilon, n)\) of the optimal uniformly \(\epsilon\)-secure partition satisfies the recurrence relation:

\[
C'(\epsilon, n) = \begin{cases} 
\infty & \text{if } n < 1/\epsilon \\
ns_n & \text{if } n \in \left[\frac{1}{\epsilon}, \frac{2}{\epsilon}\right] \\
\min_{1 \leq j \leq \frac{n-1}{\epsilon}} \left\{ \min \left( C'(\epsilon, n - j) + js_n \right) \right\} & \text{otherwise.}
\end{cases}
\]

Algorithm 2.5 shows a dynamic program for the function \(f\) which represents a lower-bound on the bandwidth of any defense that can achieve \(\epsilon\) non-uniform security in a closed-world experiment on static websites with sizes \(s_1, \ldots, s_n\) in time \(O(n^2/\epsilon)\). The dynamic program for computing uniform security lower bounds is similarly derived from the above recurrence relation.

2.4.2 From closed to open world

In this section, we show how to use closed-world experimental results to compute open-world security of defenses and open-world performance of attacks. This makes attack and defense evaluation simpler: researchers need only perform closed-world experiments to predict open-world performance.

In an open-world attack, the defender selects a website, \(W\), according to some probability distribution and generates a trace, \(T_D^D\), corresponding to a visit to that website using some defense, \(D\). The attacker’s goal is to determine whether \(W = w^*\), where \(w^*\) is a particular website of interest. (It is easy to generalize this definition to situations with multiple websites of interest). In this setting, the distribution of the random variable \(W\) corresponds to the popularity of different websites among the population of users being monitored in the attack. So, for example, if the fingerprinting attacker is a government monitoring citizens Tor usage, then \(W\) would be distributed according to the popularity of websites among that nation’s Tor users. Any closed-world attack...
Algorithm 2.5 Algorithm to compute a lower bound on the bandwidth of any offline non-uniformly $\epsilon$ secure fingerprinting defense against $A_S$ attackers.

\[\begin{align*}
\text{function} \ A_S\text{-min-cost}(n, \epsilon, \{s_1, \ldots, s_n\}) \\
\text{Array} \ C[0\ldots n\epsilon, 0\ldots n] \\
\text{for} \ i = 0, \ldots, n \epsilon \ \text{do} \\
\quad C[i, 0] \leftarrow 0 \\
\text{end for} \\
\text{for} \ i = 0, \ldots, n \ \text{do} \\
\quad C[0, i] \leftarrow \infty \\
\text{end for} \\
\text{for} \ i = 1 \rightarrow n \ \text{do} \\
\quad \text{for} \ j = 1 \rightarrow n \epsilon \ \text{do} \\
\quad\quad C[j, i] = \min_{1 \leq \ell \leq i-1} [(i - \ell)s_i + C[j - 1, \ell]] \\
\quad\text{end for} \\
\text{end for} \\
\text{return} \ C[n\epsilon, n] \\
\text{end function}
\end{align*}\]

can be used to construct an open-world attack by selecting websites $w_2, \ldots, w_n$ and building a closed-world classifier, $A$, on $w^*, w_2, \ldots, w_n$. The open-world classifier is defined as $C(t) = 1$ iff $A(t) = w^*$.

We can compute the false positive rate of this open-world attack as follows. Let $p^* = \Pr[W = w^*]$ and $p_i = \Pr[W = w_i]$ for $i = 2, \ldots, n$. We can obtain estimates for $p^*, p_2, \ldots, p_n$ from public sources, such as the Alexa “Page-Views per Million” database. Let $R_n$ be the average success rate of $A$ in the closed world, i.e.,:

$$R_n = \frac{\Pr[A(T^D_{w^*}) = w^*] + \sum_{i=2}^{n} \Pr[A(T^D_{w_i}) = w_i]}{n}$$

Note that $R_n$ is the standard performance metric used in closed-world evaluations. For simplicity, we will assume that $\Pr[A(T^D_{w^*}) = w^*] = R_n$. We also assume that, whenever $A$ misclassifies a trace, there is a $1/n$ chance that it misclassifies the trace as $w^*$, i.e., that $\Pr[A(T^D_W) = w^* \mid W \neq w^* \wedge A(T^D_W) \neq W] = 1/n$. Essentially, these two assumptions are equivalent to assuming that $w^*$ is not particularly difficult or easy for $A$ to recognize. With these assumptions, we can compute $C$’s false-positive rate:
2.4. THEORETICAL FOUNDATIONS

\[ \text{FPR}(C) = \Pr[C(T^D_W) = 1| W \neq w^*] \]
\[ = \sum_{w \neq w^*} \frac{\Pr[W = w] \Pr[C(T^D_w) = 1]}{1 - p^*} \]
\[ = \sum_{w \neq w^*} \frac{\Pr[W = w] \Pr[A(T^D_w) = w^*]}{1 - p^*} \]
\[ = \frac{n}{1 - p^*} \sum_{i=2}^{n} \Pr[W = w_i] \Pr[A(T^D_{w_i}) = w^*] \]
\[ + \left(1 - \sum_{i=2}^{n} \Pr[W = w_i] \right) \frac{1}{n(1 - p^*)} \]
\[ = \frac{1 - R_n}{n(1 - p^*)} \sum_{i=2}^{n} p_i + \frac{1}{n(1 - p^*)} \left(1 - \sum_{i=2}^{n} p_i \right) \]

With the same assumptions, the true positive rate of \( C \) is

\[ \text{TPR}(C) = \Pr[C(T^D_W) = 1| W = w^*] = R_n \]

The choice of the websites \( w_2, \ldots, w_n \) used to build \( A \) will affect the performance of \( C \) in the open world. The choice of websites affects the false-positive rate in two ways: (1) choosing less popular websites tends to increase the false-positive rate since it decreases \( \sum_{i=2}^{n} p_i \), and (2) choosing more similar websites increases the false-positive rate by reducing \( R_n \). The choice of websites affects the true-positive rate only through \( R_n \). Cai et al., showed that the Alexa top 100 websites were about as similar as 100 randomly chosen websites [10], i.e., that the most popular websites are not particularly similar to each other. Thus it is generally a good strategy to choose \( w_2, \ldots, w_n \) to be the most popular websites other than \( w^* \).

Similarly, the number, \( n \), of websites used to build \( A \) affects the false-positive rate in two ways: (1) increasing \( n \) tends to increase the false positive rate by lowering \( R_n \), and (2) increasing \( n \) tends to decrease the false-positive rate since it increases \( \sum_{i=2}^{n} p_i \). Increasing \( n \) can only decrease the true-positive rate.

Thus we can tune the false-positive and true-positive rates of \( C \) by varying \( n \). Small \( n \) will have large true- and false-positive rates. Increasing \( n \) will reduce both the false- and true-positive rates. By varying \( n \), we can generate the receiver operating curve (ROC) of \( C \).

In the real world, visits to \( w^* \) may be rare. In this case, false-positive rate can be a misleading metric. A classifier with a low false-positive rate may still be useless if true positives are so rare that they are overwhelmed by false positives. Therefore, we also report true-discovery rates for the open-world attack and defense evaluations. Given an open-world classifier, \( C \), its true-discovery
rate is defined as:

\[
\text{TDR}(C) = \Pr[W = w^*|C(T^D_W) = 1].
\]

Intuitively, the true-discovery rate is the fraction of alarms that are true alarms. The true-discovery rate can be computed from the false-positive and true-positive rates as follows:

\[
\text{TDR}(C) = \frac{\Pr[W = w^*]}{\Pr[W = w^*] \cdot \text{TPR}(C) + \Pr[W \neq w^*] \cdot \text{FPR}(C)} = \frac{p^*R_n}{p^*R_n + \frac{1-R_n}{n} \sum_{i=2}^{n} p_i + (1 - \sum_{i=2}^{n} p_i) \frac{1}{n}}
\]

2.4.3 Evaluating CS-BuFLO in the context of the optimal defense

For our experiments, we collected traffic from the Alexa top 200 functioning, non-redirecting web pages using four different defenses: plain SSH, Tor, CS-BuFLO with the CTSP padding and early termination, and CS-BuFLO with CPSP padding and early termination. In addition, to evaluate the effect of different padding schemes of CS-BuFLO, we collected smaller datasets of CPST and CTST configured CS-BuFLO pageloads. We collected 20 network traces generated by the loading of each URL, clearing the browser cache between each page load. These traces were collected in a round-robin approach. Therefore, there were about 5 hours between the collection of each network trace. The network traces were collected by four different computers running Ubuntu Linux (9.10 - 11.10), Firefox 3.6.23, Tor 0.2.1.30, and the Polipo HTTP proxy. The OnloadNotify plugin was used when CS-BuFLO pages were loaded. Network traces were recorded using tshark. For SSH pageloads, OpenSSH5.3-p1 was used and for Tor experiments, the default torrc configurations were used. SSH tunnels were created between two machines within the same network.

To evaluate the security of each defense, the network traces were used as input to the VNG++ classifier [9], Panchenko classifier [8], and the DL-SVM classifier [10]. Stratified 10-fold cross validation was used for training and testing the attack classifiers.

Security and overhead comparison. Figure 2.3 shows the level of security provided by the CS-BuFLO, SSH, and Tor defenses against three different attacks, as the number of web pages the attacker needs to distinguish increases. We find that the CS-BuFLO defense (in any configuration) performs better than Tor or SSH. Figure 2.4 plots the costs of SSH, Tor, and CS-BuFLO with CTSP and CPSP padding. SSH has almost no overhead, and Tor’s overhead is about 25% on average. CS-BuFLO with CPSP has an average overhead of 129%, CTSP has average overhead 180%. Thus CS-BuFLO’s improved security does come at a price.

Comparisons with Theoretical Bounds. Figure 2.5 shows the results of our evaluations of the overheads of the CS-BuFLO, Tor, SSH, and BuFLO defenses against the theoretical lower bounds and Figure 2.5(a) shows the results of our empirical evaluation of CS-BuFLO, Tor, and SSH on n = 120 sites and using the DL-SVM attack to estimate security. Our evaluation reveals a
large gap between the bandwidth of CS-BuFLO and the derived lower-bound. However, we see in Figure 2.5(c) that CS-BuFLO in the CTSP padding mode is over six times closer to the trade-off lower-bound than Tor for 200 sites, and is the most efficient scheme across all evaluated defenses.

Figure 2.5(b) presents the results of our empirical evaluation of CS-BuFLO, Tor, and SSH on $n = 120$ websites, using the Panchenko classifier to estimate security. We can see that CS-BuFLO offers performance in the same general range as the BuFLO configurations, but has slightly worse security in our experiments. Figure 2.5(d) shows that all but one BuFLO configuration get closer to the trade-off lower-bound curve than CS-BuFLO, Tor, and SSH (SSH is omitted from the graph because its ratio to the lower bound was never less than 400). This figure also highlights a difference between the DL-SVM and Panchenko attacks. In the DL-SVM results shown in Figure 2.5(c), Tor and SSH diverge from CS-BuFLO. In the Panchenko results in Figure 2.5(d), Tor and CS-BuFLO appear to be equally close to the lower bound.

2.5 The Provably Secure Glove Defense

Based on the results of our comparative and theoretical study of website fingerprinting defenses, we design a provably secure fingerprinting defense – Glove. Glove demonstrates that efficient and
secure website fingerprinting defenses are possible. The fundamental idea behind Glove is simple: although web pages vary widely in size and structure, they can be clustered into large groups of highly similar web pages. A defense system therefore need add only a small amount of cover traffic to make all the pages in a cluster indistinguishable to an attacker. When a user loads a web page using such a defense scheme, the attacker can identify the cluster to which the page belongs, but gains no additional information about which page within that cluster the user is loading.

Glove consists of an offline training phase and an online defending phase. During the training phase, Glove collects traces of popular web pages and clusters the pages by their network-level features. Then Glove computes, for each cluster, a transcript of packet sizes and timings that it replays, in the defending phase, whenever a user loads one of the pages in that cluster.

Glove can be viewed as an extension of the main idea behind efficient traffic morphing [57]. During the training phase, the morphing system learns the distribution of packet sizes of two websites and computes a transformation matrix that can efficiently convert one distribution into the other. During the defending phase, it uses the matrix to fragment or pad packets generated by the first website so that the resulting sizes match the distribution of the second website. Glove extends this offline-online approach to cover all traffic features, not just packet sizes. Glove can also be viewed as simply optimizing for the common case in BuFLO based defenses. BuFLO based defenses send packets between the proxy end-points at fixed schedules that reveal little information about which website is being loaded. Glove optimizes this approach by using prior knowledge about popular websites to select a packet schedule that uses less bandwidth but still hides the identity of the website. For websites that Glove does not have prior knowledge, it falls back to BuFLO type defenses. Therefore it is recommended that both systems work in conjunction with each other, rather than independently.

Glove is designed with two key goals in mind.
1. **Strong Theoretical Foundations:** Previous work on fingerprinting defenses have evaluated security of defenses by testing them against known attacks. Unfortunately, this approach can only show that a defense is ineffective, but cannot prove that a defense is secure. In this section, we present Glove, a defense scheme that is designed to defeat an ideal attacker, who can not distinguish two websites if and only if they generate the same sequence of network traffic observation. Thus the security provided by our defense is attack-independent, making Glove a provably secure defense – *i.e.*, because it gives an upper bound on the success rate of any attack, regardless of its attacking method.

2. **Reducing Overhead Costs:** The gap between the performance of existing defenses and the theoretical lower bounds derived in Section 2.4 suggests that, if a defense remembers information about websites seen in the past, it may be able to insert the cover traffic in a smart way that incurs little bandwidth overhead. We build Glove based on these insights.

Figure 2.5: Theoretical lower-bounds: Comparing existing defense overheads. Non-uniform lower bounds on bandwidth ratio, as a function of the security parameter, $\epsilon$, and specific trade-off points of the systems evaluated. SSH is omitted from Figure 2.5(d) because its ratio to the lower bound was always greater than 400.
2.5.1 The Glove defense

The network traffic generated by loading a web page consists of a sequence of packets, which we call a trace. Packets in a trace can be divided into two categories: request packets that transmitted from a client to a web server, and response packets that are directed to the client. From a different point of view, we can say the contents of a web page are covered by its trace. We now define super-trace as follows:

**Definition 2 (Super-Trace).** We say that a trace $S$ is the super-trace of traces $t_1, \ldots, t_n$ iff each trace $t_i$ may be transformed into trace $S$ by some sequence of the following actions:

- **Inserting:** Inserting request or response packets of any size.
- **Merging:** Merging consecutive request or response packets.
- **Splitting:** Splitting a packet into a set of smaller sized packets such that the sum of their sizes is maintained.
- **Delaying:** Increasing the delay between the transmission of a response packet and its succeeding request packet, and vice-versa.

Clearly, from Definition 2 we can see that, trace $t_i$ of web page $w_i$ can be replaced by super-trace $S$ (at the cost of additional bandwidth and latency overhead). This is because $S$ has enough data packets to cover the contents in $w_i$, while maintaining the dependencies between requests and responses in $t_i$.

To protect $n$ websites $w_1, \ldots, w_n$, Glove does two things in its training phase: First, it divides the traces of these websites into $k$ clusters. Second, for all the traces within a cluster, Glove computes a super-trace. In its defending phase, Glove plays the same trace $S_c$ whenever a web page in cluster $c$ is loaded, thus generating the same observation to an attacker. Glove therefore meets the information theoretic definitions of security as defined in Section 2.4. For example, let $C$ be the smallest cluster among all $k$ clusters (in terms of number of websites contained). Because loading all web pages in $C$ yields the same observation to an attacker $A$, the probability that $A$ can correctly guess which web page is loaded is $\frac{1}{|C|}$. Since $C$ is the smallest cluster, Glove is a uniformly $\epsilon$-secure defense, where $\epsilon = \frac{1}{|C|}$. In this example, Glove can also be described as a non-uniform $\epsilon$ secure defense, where $\epsilon = \frac{k}{n}$.

2.5.1.1 Clustering webpages

At a high level, Glove clusters webpages using the $k$-medoids clustering algorithm on the Dynamic Time Warping (DTW) based distance matrix (computed on input webpage traces).

**Finding representative traces.** In reality, not all webpages are static and network conditions change from time to time. As a result, traces may vary each time a webpage is loaded. Therefore, before clustering webpages, we need to choose a “representative” trace for each webpage, i.e., the
2.5. THE PROVABLY SECURE GLOVE DEFENSE

trace that is likely to be most easily (with the lowest bandwidth) transformed into other traces generated by the same page. We accomplish this by loading each webpage \( n \) times (recording \( n \) traces for each site) and then computing pair-wise Damerau-Levenshtein edit distances. Given the pair-wise edit-distances, we use the trace \( t \) with the minimum average distance to others as the representative trace for the web-page.

**k-medoids clustering.** A network trace can be viewed as a time-series, with packets (events) occurring at certain times. We use this observation in our clustering technique. We compute the Dynamic Time Warping (DTW) \(^{[67]}\) distances between every pair of representative traces. Once this pairwise distance matrix is computed, we use the \( k \)-medoids \(^{[68]}\) algorithm to group similar webpages into a pre-determined number of clusters. The number of clusters determines the security and overhead of the Glove defense.

We selected the DTW edit-distance metric over other metrics after simulations revealed that, on average, the clusters generated by DTW had lower overhead super-traces. This is because DTW implicitly takes into account the time between packets while computing edit distances – an important factor when considering that supertraces need to maintain the inter-packet time dependencies between requests and responses in their constituent traces. In our implementation, the cost of a cluster configuration is the lower-bound on bandwidth overhead. This can be computed as: 

\[
\sum_{i}^{k} |c_i| (\max(\text{req}_j \in c_i) + \max(\text{res}_j \in c_i)).
\]

Here, \( c_1, \ldots, c_k \) are the \( k \) clusters and \( \text{req}_j \), \( \text{res}_j \) denote the number of request and response bytes of the \( j^{th} \) site. The idea is that any super-trace of a group of traces must contain at least as many request and response bytes as its constituent traces with the most number of request and response bytes. For clustering, the \( k \)-medoids algorithm was selected due to its ability to account for custom distance metrics between points (unlike \( k \)-means and other clustering methods which assume a euclidean space).

### 2.5.1.2 Computing super-traces

The super-trace of a cluster is a single trace which covers all traces contained in that cluster. If all webpages are static, a defense that plays this supertrace while loading any webpage within the cluster, effectively hides all information about the page being loaded (except the cluster it belongs to). Unfortunately, webpages are dynamic. Therefore, we compute a supertrace which aims to conservatively cover a large (tunable) percentage of all traces that any of its constituent webpages might generate. To do this, we use the heuristic demonstrated in Algorithm \(^{[2,6]}\) to approximate the minimum bandwidth super-trace for each cluster. The following notation is used:

- **Minimum site coverage (\( \mu_{\text{min}} \)):** This parameter determines the minimum number of traces of each webpage to be covered by the super-trace. The parameter \( \mu \) denotes the average coverage of all pages. Larger \( \mu_{\text{min}} \) values provide more resistance to the dynamicity of webpages, often times at the cost of larger overheads (not always).

- **Bandwidth-Latency tuner (\( \tau \)):** This parameter allows us to tune the defense to produce super-traces that optimize some combination of bandwidth and latency overheads. The lower the
value of $\tau$, the lower the latency overhead (at the cost of bandwidth), and vice versa. The range of $\tau$ is 1 to 100.

- $T$ is the set of input traces that the super-trace is computed over. This is initially $\emptyset$. $ST$ denotes the currently computed super-trace. $R_i$ is the set containing all the recorded traces of site $i$. $cov_{min}$ is the index of the site which is least covered by the current $ST$. This value may be initialized randomly.

- $F$ denotes the current frontier packet of each trace in $T$ and $len_i$ denotes the number of packets in the $i^{th}$ trace in $T$.

- The function $\text{Find-Direction}$ returns +1 if more than $\frac{1}{6}$ up-stream packets, else returns -1. Function $\text{Find-Time}$ returns the $\tau^{th}$ percentile time of frontier packets in direction $P_D$. $\text{Find-Size}$ returns the maximum packet size in the frontier with time $\leq P_T$ and direction $= P_D$. Finally, function $\text{Update-Frontiers}$ updates the frontier of each trace ($F_i$) to the last packet not covered by the current $ST$.

The algorithm is simple. The super-trace is computed over a set of input traces. In each iteration we add a trace from the least covered site into this input, until all sites have satisfied the minimum coverage parameter $\mu_{min}$.

For each of these input traces, a counter (starting at the first packet) indicating the current frontier is maintained. We count the number of frontier packets in each direction. If more than $\frac{1}{6}$ of the packets are up-stream packets, we add an up-stream packet to the super-trace, otherwise, we add a down-stream packet. We use the parameter $\frac{1}{6}$ after performing experiments that revealed the average ratio of up-stream to down-stream packets to be $\frac{1}{6}$. This newly added packet is pushed to the network at the time of the $\tau^{th}$ percentile frontier packet (assuming that frontier packets are ordered by time) in the chosen direction. The size of the newly added packet is the maximum size (rounded to the nearest 50 bytes) of all the frontier packets in the chosen direction.

The above process is repeated until the frontier of all the input traces have passed their final packet. The final trace $ST$ is guaranteed to cover at-least $\mu_{min}%$ of the traces of each site (although, in practice the average coverage turns out to be far higher).

### 2.5.2 Evaluation methodology and results

For our evaluation of Glove, we collected 50 traces each from the Alexa top 500 functioning, non-redirecting web pages. The browser cache was cleared between page-loads. Next, each of the 50 traces for each site were ranked by their representativeness. Finally, $k$-medoid clustering was performed to identify websites with similar representative traces. The number of clusters ($k$) was varied in the range of 16 and 250, making Glove a provably $\epsilon$-non-uniformly secure fingerprinting defense, where $\epsilon$ varied between 0.032 and 0.5. After clustering, sets of super-traces were generated for each of the clusters while varying $\mu_{min}$ and $\tau$. Finally, statistics corresponding to expected site coverage, bandwidth, and latency ratios were computed.
2.5. **THE PROVABLY SECURE GLOVE DEFENSE**

Algorithm 2.6 Algorithm to compute the super-trace of a Glove cluster

```plaintext
function INPUT-GEN(\(\mu_{\text{min}}, \tau, \text{covmin}, T, \{R_1, \ldots, R_n\}\))
  if \(\text{coverage}_{\text{covmin}} < \mu_{\text{min}}\) then
    \(T \leftarrow T \cup t\), where \(t \in R_{\text{covmin}}\).
    \(ST \leftarrow \text{compute-st}(\tau, T)\)
  else
    return \(ST\)
  end if
end function

function compute-st(\(\tau, T\))
  \(ST \leftarrow \emptyset\)
  \(F \leftarrow \{1, \ldots, 1\}\)
  while \(F \neq \{\text{len}_1 + 1, \ldots, \text{len}_m + 1\}\) do
    \(P_D \leftarrow \text{Find-Direction}(T)\)
    \(P_T \leftarrow \text{Find-Time}(\tau, T, P_D)\)
    \(P_S \leftarrow \text{Find-Size}(T, P_T, P_D)\)
    \(ST \leftarrow ST \cup (P_D, P_T, P_S)\) \(\triangleright\) Add a packet of size \(P_S\) in direction \(P_D\) and at time \(P_T\) to the super-trace
    \(F \leftarrow \text{Update-Frontiers}(ST, T)\)
  end while
end function
```

**Overhead and security trade-off.** Figure 2.6 compares the trade-off between overhead-security trade-offs of Glove, BuFLO, and CS-BuFLO. From the evaluated defenses, Glove is the only defense that provides information theoretic security against any attacker, while the BuFLO and CS-BuFLO defenses can only be evaluated against the Panchenko and DL-SVM classifiers, respectively. The implication here is that while the BuFLO and CS-BuFLO trade-offs might vary depending on the attacker, the Glove trade-off holds against any attacker. From the figure, it is clear that Glove provides better trade-off costs than CS-BuFLO (vs. DL-SVM, for any security level) and BuFLO (vs. Panchenko, when attack accuracy \(\leq 10\%\)). When security is less critical (attack accuracy \(> 10\%\)), however, Glove and BuFLO (vs. Panchenko) provide similar trade-offs.

**Tunability of Glove.** In Figure 2.7, we demonstrate the effect of varying the \(\mu_{\text{min}}\) and \(\tau\) parameters of Glove (on its bandwidth and latency ratios). In particular, the figures show that varying \(\mu_{\text{min}}\) to increase resistance to dynamic content in web pages has little effect on the bandwidth overhead of the defense, while increasing its latency overhead (when \(\tau = 99\% - \text{i.e., }\), the defense is optimized for minimizing bandwidth costs). Further, we see that \(\tau\) allows user experience tuning – \(\text{i.e.,}\), reducing \(\tau\) causes a significant drop in latency overheads (increasing the usability of the defense) at the cost of increased bandwidth.

2.5.3 **Glove in the real-world**

Glove demonstrates a promising new approach towards building efficient fingerprinting defenses. In particular, Glove:

**Provides information-theoretic security guarantees.** This is achieved by computing a single super-trace for each computed cluster and playing this each time a page contained in the cluster is
loaded. As a result, in the absence of prior (or, outside) knowledge, any (current or future) website fingerprinting attackers success rate is limited by the size of the computed cluster.

**Has good overhead-security trade-off.** The results illustrated in Figure 2.6 demonstrate the validity of our conjecture that using prior information about the structure of a web page to add cover traffic conservatively yields better website fingerprinting defenses.

**High tunability.** Unlike previous defenses, Glove allows users to tune their browsing experience by adjusting the $\tau$ parameter. Lower $\tau$ values reduce latency at the cost of bandwidth, and vice-versa. This is especially useful in scenarios where it is likely the case that bandwidth costs are inconsequential, while even moderately increased user experienced latency may result in poor usability of the defense.

However, Glove also has two main limitations that may hamper its adoptability in the real-world
2.6. CONCLUSIONS

– high infrastructure requirements and inability to deal with highly dynamic content.

**Infrastructure requirements.** Glove greatly reduces its overhead by utilizing prior knowledge of web page structures. This requires the **Glove infrastructure** to be able to collect traces of defended web pages, cluster these pages, and compute corresponding super-traces. These tasks are computationally intensive, making them infeasible for standard server side proxy nodes to perform on a regular basis. Instead, using a powerful dedicated central node to compute clusters/super-traces and distribute them to Glove proxies (e.g., via Tor bridges – since the security of Glove is independent of the secrecy of the computed super-traces) is more efficient. However, such a node may not be easily available in the real-world. In the absence of such nodes, Glove has to fall back on less powerful server side proxies which take as input a list of URLs (from the client) and returns a single super-trace (to be played when the client loads a page in the input URLs). However, in these cases, while Glove retains its information-theoretic security guarantees, it is no longer able to use prior knowledge of web page structures to provide low overheads. To circumvent these problems, one may consider developing distributed clustering and super-trace algorithms for use with Glove.

**Effect of dynamic content.** Two types of dynamicity affect the Glove defense. First, the information-theoretic guarantees provided by Glove hold only when $\mu_{\text{min}} = 100\%$. This, however, is possible to guarantee only when web pages do not contain dynamic content (i.e., JS, AJAX, etc.). Second, when the structure of a Glove defended web page changes significantly, its trace may change to a large enough degree that less than $\mu_{\text{min}}\%$ of its page loads are covered by its current super-trace. In this case, re-clustering/super-tracing of all defended pages may be in order. While our observation is that this is rare, its occurrence does result in the need for the Glove infrastructure to occasionally perform this re-clustering, super-tracing, and distribution.

### 2.6 Conclusions

In this chapter we presented CS-BuFLO – a congestion-sensitive variant of the previously proposed BuFLO website fingerprinting defense, theoretical foundations that enable bootstrapping and evaluation of provably secure website fingerprinting defenses, and Glove – a provably secure defense based on the proposed theoretical foundations.

CS-BuFLO offers a real world implementation of a high-security, moderate-overhead solution to website fingerprinting attacks. Compared to SSH and Tor, it achieves a better security/bandwidth trade-off, i.e. it uses its bandwidth efficiently to provide extra security. The padding schemes developed for CS-BuFLO, along with browser-coordination and our early-termination algorithm, can improve security with less overhead than previous stream padding schemes. Interestingly, we also found that padding from one end of a connection can sometimes be an efficient way to hide information about the data sent from the other side of the connection. We find that additional client padding does improve security by obscuring the size of the final object requested by the client. CS-BuFLO's security/overhead trade-off is in the same range as the estimates Dyer obtained for BuFLO in their simulations. For example, Dyer et al. reported that, in one configuration of
BuFLO, bandwidth overhead was 200% and the Panchenko SVM had an 24.1% success rate on 128 websites. We found that CS-BuFLO with CTSP padding had an overhead of 180% on 120 websites, and that the Panchenko SVM had a success rate of 23.4%. Most importantly, CS-BuFLO's congestion-sensitivity means that, in a real deployment, it is likely to have even lower bandwidth overheads.

Our theoretical foundations clarify the limits of website fingerprinting defenses. It establishes efficiency bounds that no defense can cross, giving an absolute benchmark for evaluating the efficiency of defenses. The lower bounds of bandwidth costs are surprisingly low, suggesting that it may be possible to build very efficient defenses. We also show that, in some contexts, randomized defenses offer no security or overhead advantage compared to deterministic defenses. This theoretical foundation also provides a framework for comparing schemes which offer different overhead and security trade-offs. Further, it allows conclusions to be drawn about open-world performance of attacks and defenses, based on their closed-world results. This greatly simplifies the experimental setup required to estimate open-world performance of attacks and defenses. Based on these foundations, we bootstrapped Glove— a provably secure website fingerprinting defense. Although Glove suffers from limits to its practicality, it is a proof-of-concept that illustrates the feasibility of low-overhead and provably secure approaches to defend against website fingerprinting adversaries.
Chapter 3

Network-level Traffic Correlation Defenses

Contents

3.1 Introduction .......................................................... 42
3.2 Evolution of Network-level Traffic Correlation Research .......... 43
3.3 Measuring Adversary Presence ....................................... 44
   3.3.1 Network-level path prediction .................................. 45
   3.3.2 Measurement methodology .................................... 48
   3.3.3 Experiments and results ....................................... 49
3.4 The Cipollino Tor Client ............................................. 56
   3.4.1 Providing attacker free circuits ................................ 57
   3.4.2 Reducing performance overhead ............................... 59
   3.4.3 Considering relay-level attacks ............................... 61
   3.4.4 Load-balancing across relays ................................ 62
   3.4.5 Considering the worst-case .................................. 63
   3.4.6 Putting it all together: The Cipollino client architecture .... 64
3.5 Conclusions .......................................................... 65

In this chapter, we focus on measuring and mitigating the threat faced by Tor users from traffic correlation attacks by network-level adversaries. These attacks may be launched by Autonomous Systems (ASes) that are in a position to observe traffic entering and exiting the Tor network. The results presented in this chapter have previously appeared as part of the following articles: [22] [23].
CHAPTER 3. NETWORK-LEVEL TRAFFIC CORRELATION DEFENSES

3.1 Introduction

Traffic correlation attacks to de-anonymize Tor users are possible when an adversary is in a position to observe traffic entering and exiting the Tor network. In a traffic-correlation attack, an adversary correlates the characteristics of traffic (e.g., packet sizes, inter-packet timings, etc.) entering and exiting the Tor network. Successfully correlating these flows results in the de-anonymization of Tor users – i.e., it becomes possible to identify the destination server being contacted by a Tor user. The existence of these attacks have been known about for over a decade [69] but as our study shows, Tor is still vulnerable. Worse yet, the recent Snowden leaks have confirmed that the NSA and GCHQ, in collusion with Internet Service Providers (ISPs), have actively been working to implement network-level attacks in the wild [70, 71, 72].

In order to launch a traffic correlation attack, an adversary needs to be able to observe network traffic on (1) the (forward- or reverse-) path between the Tor user and the entry (relay) to the Tor network and (2) the (forward- or reverse-) path between the exit (relay) from the Tor network to the destination server. Figure 3.1 illustrates this condition. Here, to de-anonymize a Tor client, an AS needs to be present on one of the solid path segments and on one of the dashed path segments.

![Network-level correlation attack pre-requisites](image)

Figure 3.1: Network-level correlation attack pre-requisites: An AS needs to be present on one of the two solid path segments – i.e., path segment A or B – and on one of the two dashed path segments – i.e., path segment C or D.

More formally, if \( P_{SRC\leftrightarrow EN} \) is the set of ASes on the forward and reverse paths between the Tor client (source) and the selected Tor entry-relay and similarly, \( P_{EX\leftrightarrow DST} \) is the set of ASes on the paths between the selected Tor exit-relay and the destination, then we say that a Tor circuit is vulnerable to de-anonymization via traffic-correlation if there is some AS \( A \) such that:

\[
A \in \{P_{SRC\leftrightarrow EN} \cap P_{EX\leftrightarrow DST}\} \tag{3.1}
\]

An AS may satisfy Equation (3.1) through passive or active means:

**Passive adversaries.** An AS may find itself in a position to launch a traffic-correlation attack simply as a result of the AS-level topology and the relationships (i.e., customer-provider or peer-peer) it shares with other ASes. In order to defend against attacks from passive adversaries, it is sufficient to have an accurate snapshot of the ASes that occur in the sets \( P_{SRC\leftrightarrow EN} \) and \( P_{EX\leftrightarrow DST} \) for each choice of entry-relay (\( EN \)) and exit-relay (\( EX \)). Given this information, a correlation attack can be avoided by simply selecting an entry- and exit-relay for which there is no AS \( A \) which satisfies Equation (3.1) (if such an entry- and exit-relay combination exists).

**Active adversaries.** Due to the dynamics and insecurities of the BGP protocol, ASes may also
actively seek to place themselves in a position to launch traffic-correlation attacks. For example, an AS may hijack or intercept traffic sent to the prefix associated with the client, entry-relay, exit-relay, or destination server. Such targeted hijacks and interceptions potentially allow adversaries to place themselves on any of the four paths illustrated in Figure 3.1. Defending against such adversaries is more challenging due to need for access to real-time control-plane data to identify AS that are likely to be hijacking or intercepting traffic. This is in addition to the snapshots of AS-level paths required for defending against passive adversaries.

In this chapter, we present our efforts to: (1) measure the threat of these attacks on Tor users and (2) defend Tor from such attacks by network-level adversaries. To measure the threat, we conduct a series of experiments under different settings to identify the number of ASes that are in a position to launch correlation attacks via active and passive approaches. In the context of defending against such attacks, there are two possible approaches: (1) Tor relays may add noise to streams passing through them, making correlation of streams entering and exiting the network infeasible (e.g., the padding approaches described in Chapter 2), or (2) The Tor client may perform relay-selection to ensure that no AS is in a position to perform a correlation attack. We focus on the latter approach.

3.2 Evolution of Network-level Traffic Correlation Research

The problem of network-level traffic correlation attacks on Tor was first considered by Feamster and Dingledine in 2004 [69]. Their study, conducted on a nascent Tor network (consisting of 33 relays), revealed that 10-30% of all circuits constructed by Tor were vulnerable to traffic correlation attacks by network-level adversaries. This work relied on algorithmic simulations based on the work of Gao [73] and Mao et al. [74]. Following this, Murdoch and Danezis [75] and Murdoch and Ziełinski [76] proposed the first practical traffic analysis attacks for network-level adversaries.

In 2009, after the Tor network grew to over 2K relays and 250K daily users, the problem was re-visited by Edman and Syverson [77]. In addition to showing that between 18-21% of all circuits generated by the Tor client were vulnerable to traffic correlation attacks when considering the asymmetric nature of Internet routing, they proposed defense heuristics for relay selection. Empirical data showed that these heuristics reduced the fraction of vulnerable circuits to 6-16%. For the purpose of inferring AS-level paths, algorithmic simulations combining the AS-level path inference algorithm from Qiu et al. [78] with Gao’s relationship inference model [73] were used. The first fully specified client-based defense – LASTor – was proposed by Akhoondi et al. [79]. While LASTor successfully mitigated the threat of traffic correlation attacks on forward-paths (i.e., source to entry-relay and exit-relay to destination), it ignored attacks on reverse-paths (i.e., entry-relay to source and destination to exit-relay). Additionally, LASTor did not account for bandwidth capacities of relays during relay selection. Accounting for relay capacity is required to ensure that no single set of relays are over-loaded by a large number of clients – therefore reducing the performance of the entire network [80]. The performance impact of their capacity-independent
relay selection was masked by using HTTP HEAD vs. full page loads to evaluate throughput. To perform path estimation, LASTor relied on path stitching techniques borrowed from iPlane Nano [81] applied to BGP control-plane feeds.

Following revelations about mass surveillance, research considering the measurement of and defenses against traffic correlation attacks received renewed attention. In 2013, Johnson et al. [82] showed via empirical evaluations that network-level correlation attacks required minimal investment from adversaries (in terms of bandwidth resources) to de-anonymize Tor clients in selected locations. In addition, their study revealed that Tor clients using certain applications (e.g., IRC and BitTorrent) were more vulnerable than others. This is due to the smaller number of exit-relays supporting connections to the corresponding (application specified) ports. For the purpose of AS-level path inference, algorithmic simulations based on the work of Qiu et al. [78] were used. Sun et al. proposed RAPTOR [21], a collection of correlation attacks that exploited BGP dynamics and path asymmetry. In particular, RAPTOR proposed and evaluated the first practical traffic correlation attacks to exploit asymmetric Internet paths and the possibility of launching BGP hijacks and interceptions to selectively de-anonymize Tor users in just 300 seconds. Highlighting the importance of using high-fidelity measurement data for measuring and defending against network-level adversaries, Juen et al. [83] performed a comparison between AS-level paths computed by Qiu et al.’s [78] algorithmic simulations and data-plane measurements (i.e., traceroutes). They found that about 20% of all inferred paths had at-least two missing ASes relative to paths measured with traceroute.

In 2016, Nithyanand et al. [22] proposed and developed Astoria– an AS-aware Tor client that considered attackers capable of exploiting forward- and reverse-paths to launch traffic correlation attacks. In addition, Astoria avoided the problems faced by LASTor by performing capacity-aware relay selection. However, like previous work, Astoria also relied on algorithmic simulations to obtain AS-level paths [41]. In a follow-up to Astoria, Nithyanand et al. [23] proposed and developed Cipollino– an AS-aware Tor client able to defend against both, passive and active AS-level adversaries. In addition to obtaining high-fidelity paths from data- and control-plane measurements, Cipollino also incorporated optimizations to reduce the latency and computational overhead of AS-aware Tor clients. Table 3.1 summarizes and compares measurement and defense research contributions in the field of network-level correlation attacks on Tor.

3.3 Measuring Adversary Presence

In this section, we measure how often ASes are in a position to perform traffic correlation attacks on Tor users. First, we detail how prediction of AS paths between a source and a destination is performed and how sets of potential attacker ASes are identified. Then we present our experiments and their results.
### 3.3. MEASURING ADVERSARY PRESENCE

<table>
<thead>
<tr>
<th>Path Prediction Approach</th>
<th>Exploits Considered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feamster &amp; Dingedine [69]</td>
<td>X</td>
</tr>
<tr>
<td>Edman &amp; Syverson [77]</td>
<td>X</td>
</tr>
<tr>
<td>Akhoondi et al. [79]</td>
<td>X</td>
</tr>
<tr>
<td>Johnson et al. [82]</td>
<td>X</td>
</tr>
<tr>
<td>Juen et al. [83]</td>
<td>√</td>
</tr>
<tr>
<td>Nithyanand et al. [22]</td>
<td>X</td>
</tr>
<tr>
<td>Nithyanand et al. [23]</td>
<td>√</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of the measurement methodologies and defense contributions of the state-of-the-art. N/A indicates that the work only made measurement contributions and did not propose a defense.

#### 3.3.1 Network-level path prediction

To accurately measure the threat from network-level adversaries, it is vital to correctly identify ASes on the paths from and to the selected entry- and exit-relays. There are three options for predicting paths between pairs of ASes:

1. **Data-plane measurements.** Data-plane path measurement tools such as traceroute allow measurement of exact paths between a source and destination host. However, this requires control of the source host, which may not always be possible (e.g., it is not possible to traceroute between the exit-relay and destination server) and has a high latency cost, making it infeasible for clients to perform on-demand.

2. **Control-plane measurements.** Paths may also be obtained via control-plane measurement infrastructure such as BGP monitors (e.g., RIPE [85], Routeviews [86]). However, they (like data-plane infrastructure) are limited by the location and peers of the BGP monitors.

3. **Algorithmic simulations.** This approach relies on simplified assumptions about Internet routing. Typically, algorithmic simulators use empirically derived AS-level topologies, inferred inter-AS relationships (e.g., customer-provider or peer-peer), and a simplified model of Internet routing policies (e.g., [84, 41]). While algorithmic simulators are able to predict AS-level paths between any pair of ASes, their accuracy compares unfavorably with paths obtained from data- and control-plane measurements. This is due to the incompleteness of AS-level topologies and the absence of ground-truth while inferring AS relationships.

To understand the impact of inaccurate path predictions, we test the accuracy of the state-of-the-art simulator [41] which relies on the Gao-Rexford routing model [84]. For our experiment, 225 pairs of exit-relay and destination ASes are taken from circuits created by loading the Alexa...
CHAPTER 3. NETWORK-LEVEL TRAFFIC CORRELATION DEFENSES

Figure 3.2: Network-level path prediction: Number of ASes over or under-estimated by the state-of-the-art algorithmic simulator.

top-100 websites using the vanilla Tor client. For each pair, a traceroute was executed from the AS containing the exit-relay (vantage points were obtained using RIPE Atlas probes). IPs from each traceroute hop were resolved to their ASes using up-to-date BGP announcement data to produce AS-level paths. These AS-level paths were compared with the AS-level paths obtained by the algorithmic simulator. Figure 3.2 shows the result of comparing measured with simulated paths. We find that straightforward application of simulation can lead to over-estimating the number of ASes present 80% of the time. Worse yet, 40% of simulated paths actually miss ASes contained in the paths, potentially leaving the client vulnerable to traffic correlation attacks.

To mitigate the impact of such inaccurate path-prediction, we reduce our dependence on algorithmic simulators by using PathCache [40] – a system that aggregates existing data- and control-plane measurements to predict paths. The main idea behind PathCache is to perform AS-level path prediction by utilizing existing publicly available measurements obtained from data-plane measurement platforms such as RIPE Atlas [87], iPlane [81], CAIDA Ark [88], and control-plane measurement platforms such as RouteViews [86], RIPE RIS [85], and others. We fall back to simulations only when a path query cannot be answered using measurement data. To understand the benefits of PathCache, we evaluate it on two criteria: (1) accuracy of predicted paths and (2) the fraction of paths where PathCache is able to answer using empirical control- and data-plane derived data (vs. algorithmic simulations).

We measure the number of ASes over- or under-estimated when compared with 225 traceroutes that were not already aggregated by PathCache. Figure 3.3 illustrates the results of this experiment. We find that PathCache is more accurate than the state-of-the-art algorithmic simulator (cf. Figure 3.2). Most importantly, with 84% of all paths having no missing ASes (no under-estimations), PathCache is less likely to under-estimate the adversary presence. This is important given our plan to integrate this measurement system with our defense.

To understand how often PathCache is able to predict paths using empirical data, we queried PathCache for paths between (1) 1,000 source ASes (100 of the most populous ASes in each of the ten countries) and the ASes of all entry-relays in the Tor network (265K path queries) and (2) between the ASes of all exit-relays in the Tor network and all the destination ASes seen in 2K
3.3. MEASURING ADVERSARY PRESENCE

Figure 3.3: Network-level path prediction: Number of ASes over or under-estimated by PathCache, when compared to exact AS-level paths obtained by traceroutes.

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Coverage (Percentage)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRC ← EN</td>
</tr>
<tr>
<td>10</td>
<td>36.6</td>
</tr>
<tr>
<td>25</td>
<td>32.1</td>
</tr>
<tr>
<td>50</td>
<td>27.6</td>
</tr>
<tr>
<td>100</td>
<td>23.2</td>
</tr>
</tbody>
</table>

Table 3.2: PathCache coverage: Percentage of paths predicted by PathCache when considering only the top N percentile of relays (by bandwidth).

web-page loads (312K path queries). The 2K web-page loads consisted of the Alexa Top 100 sites and 100 sensitive web-pages [89] for each of ten countries – Brazil (BR), China (CN), Germany (DE), Spain (ES), France (FR), England (GB), Italy (IT), Russia (RU), Ukraine (UA), and the United States (US). Table 3.2 shows the percentage of paths that were predicted by PathCache using empirical data. Here we see that PathCache is able to achieve reasonable coverage when considering high capacity entry- and exit-relays (34-36%). This implies a higher accuracy of paths predicted for organically generated Tor circuits, as the Tor client will tend to use these higher capacity relays.

In Figure 3.4 we see the per-country breakdown of the fraction of path requests satisfied by PathCache. Interestingly, we see BR and CN in particular having a small fraction of paths between their 100 AS sources and the Tor entry-relays. We speculate that this is due to blocking of communication with Tor entry-relays in these countries. This prevents traceroutes (that PathCache uses as a basis for path prediction) from successfully traversing paths from client ASes in these countries to Tor entry relay ASes. We find that depending on client location, PathCache is able to answer between 15-50% of all queries issued to it by the Tor client. Importantly, the paths returned for these queries are unlikely to under-estimate the presence of an AS. Additionally, coverage increases when considering higher capacity Tor relays. These factors make it a good alternative to relying on simulations for path prediction. When PathCache is unable to answer a query due to the absence of empirical data, we rely on algorithmic simulations.
CHAPTER 3. NETWORK-LEVEL TRAFFIC CORRELATION DEFENSES

Figure 3.4: PathCache coverage: Per-country breakdown of $SRC \leftarrow EN$ and $EN \leftarrow SRC$ paths predicted by PathCache.

3.3.2 Measurement methodology

We use a combination of live experiments using VPN vantage points and simulations to understand the threat to Tor in practice, at scale, and over time. In our experiments, we consider the fact that due to regional differences in AS-level topologies, Tor clients in different regions face varying levels of vulnerability. Therefore, we consider Tor clients located in ten countries: Brazil (BR), China (CN), Germany (DE), Spain (ES), France (FR), England (GB), Italy (IT), Russia (RU), Ukraine (UA), and the United States (US). This list of locations is an intersection of the countries with the largest number of Tor users [5] and the countries ranking the lowest on the Freedom House Internet freedom rankings [90].

Live experiments with VPNs. In each live experiment, we use Crawler Incantatus [91] (a Selenium based web-crawler) and the Tor client to load web-pages from within each country, using a commercial VPN service.

Simulating Tor behavior with TorPS. VPN vantage points only provide us a limited view of the threat in each country – i.e., the threat faced by the Tor client located in a single AS. Therefore, in our simulations we considered Tor clients located in 100 of the most popular (in terms of end-users [92]) ASes in each country. To simulate Tor circuit construction, we use the Tor Path Simulator (TorPS). TorPS is a realistic Python-based Tor simulator which uses published Tor server-descriptors and consensuses from the CollecTor project [6] to model the Tor client. Given (1) the set of server descriptors corresponding to the period of the experiment and (2) the set of streams generated by the user (each stream consists of a set of IP addresses, ports, and connection request times), the TorPS simulator constructs circuits for each connection request within the stream, according to a chosen client model (in our case the vanilla Tor client). This allows us to predict the relays that would have been selected by the Tor client for a given network state.

Marking vulnerable circuits. In the remainder of our experiments, we consider a circuit constructed by a Tor client to be vulnerable if the set of ASes $A$ in Equation 3.1 is non-empty. Here, we use the PathCache framework to identify the ASes on $P_{SRC\leftrightarrow EN}$ and $P_{EX\leftrightarrow DST}$.

Each experiment was executed in the live or simulated setting in each of the ten listed countries.
We also used TorPS with historical states of the Tor network obtained from the CollecTor project to analyze the evolution of the threat from network-level adversaries since 2010. Logs were maintained to track the circuits established by the Live or Simulated Tor client for each experiment.

### 3.3.2.1 Workload models

When the Tor client selects relays for a circuit, it may only select exit relays that can transport the type of traffic to be sent over the circuit (e.g., some exits restrict commonly abused ports such as port 25 – SMTP). As a result, the vulnerability of the Tor client can depend on the applications used by the Tor user. We consider two different client workloads described below.

**Web workload.** For each experiment using the web workload, 200 websites were loaded by the Tor client. The list of 200 websites were dependent on the client location – i.e., comprised of the local Alexa Top 100 sites and 100 country-specific sensitive (likely to be blocked or monitored) web-pages obtained from the Citizen Lab testing list repository. In the case of simulated Tor clients, streams that were used as input to the TorPS simulator were constructed using the IPs and ports observed in the live experiments.

**Mixed (application) workload.** For each experiment considering a mixed application workload, we considered clients that used Tor for a mix of Web, P2P (BitTorrent), e-mail and IRC chat for an hour long period. The purpose of these experiments was to understand if the security of the Tor client was affected when users required connections to non-HTTP(S) ports.

### 3.3.3 Experiments and results

**E1: Measuring vanilla Tor’s vulnerability to AS-level adversaries (web workload).** In this experiment we measure the fraction of vulnerable circuits constructed by the vanilla Tor client and the fraction of websites that use one of these vulnerable circuits. The results, for each of the ten countries, are illustrated in Figure 3.5. The experiments were conducted using a VPN vantage point in each of the ten countries, while using the web workload workload.

![Figure 3.5](image-url)

Figure 3.5: Network-level correlation attacks: Current vulnerability (web workload). Per-country breakdown of fraction of vulnerable websites and circuits [E1].
Observation: While only 31% of the circuits constructed by the Tor client are vulnerable to AS-level adversaries, we find that due to aggressive circuit re-use and concentration of websites in a few ASes, that a larger fraction (58%) of all websites loaded by the clients end up using a vulnerable circuit.

E2: Measuring current vulnerability to AS-level adversaries (mixed workload). In this experiment we measure the fraction of vulnerable circuits constructed by the vanilla Tor client when used for a mix of loading webpages, sending email, communicating *via* IRC chat, and downloading files using BitTorrent – *i.e.*, the mixed workload. The results for each of the ten countries are illustrated in Figure 3.6. The TorPS simulator and a user workload based on streams generated by the above applications were used. 100 of the most populous (in terms of end-users) ASes \[92\] in each of the ten countries were selected as Tor client locations.

![Figure 3.6: Network-level correlation attacks: Current vulnerability (mixed workload). Per-country break down of fraction of circuits found to be vulnerable with the Web and mixed workloads. \[E2\].](image)

Observation: We find that although the average vulnerability of mixed application clients (30%) in the countries is similar to web-only clients (31%), the average vulnerability of clients in DE, FR, and UA are most affected by considering mixed application traffic. This implies that the few exit-relays that allow communication over non-HTTP(S) ports enable at-least one AS to perform a traffic correlation attack, given clients located in these countries.

E3: Measuring historical vulnerability to AS-level adversaries (web workload). In this experiment we measure the fraction of vulnerable circuits constructed by the vanilla Tor client working with the web workload, while considering the changing landscape of the Tor ecosystem between 2010 and 2015. In each country we consider clients located in the 100 most populous ASes \[92\]. Figure 3.7 illustrates our results. Here, we show the average fraction of vulnerable circuits for clients in all 1000 ASes, the country whose 100 ASes had the least average vulnerability (FR), and the country whose 100 ASes had the highest average vulnerability (CN).

Observation: Most countries have an average of 25-45% of their circuits remaining vulnerable to AS-level attackers. China is an exception with an average of 50-60% of their circuits remaining vulnerable. Further, in spite of the addition of nearly 6K new relays in the Tor network (since 2010), the average threat from AS-level adversaries has grown – from 38% of all circuits being
3.3. MEASURING ADVERSARY PRESENCE

Figure 3.7: Network-level correlation attacks: Historical vulnerability (web workload). The average, current minimum, and current maximum fraction of vulnerable circuits constructed by vanilla Tor when considering web clients located in each of ten countries and the Tor network between 2010 and 2015 [E3].

E4: Measuring historical vulnerability to AS-level adversaries (mixed workload). Here, we use the same settings as experiment E3, only changing the workload – i.e., while E3 calculated the fraction of vulnerable circuits for users loading 200 webpages in each country, here we consider users who perform a variety of non-http(s) related communication via Tor – e.g., IRC, email, BitTorrent, etc.. The results are illustrated in Figure 3.8.

Figure 3.8: Network-level correlation attacks: Historical vulnerability (mixed workload). The average, current minimum, and current maximum fraction of vulnerable circuits constructed by vanilla Tor when considering mixed workload clients located in each of ten countries and the Tor network between 2010 and 2015 [E4].

Observation: We find that the threat faced by clients that use Tor for a mix of non-Web applications is currently slightly lower than web-only Tor clients, in general. However, the threat has been growing at a significantly faster rate. We see in the last five years that the average threat (in terms of vulnerable circuits constructed in the course of our experiments) has increased from 21% to 35%.

Discussion. Our results indicate that the threat from de-anonymization by AS-level adversaries
CHAPTER 3. NETWORK-LEVEL TRAFFIC CORRELATION DEFENSES

is high, regardless of client location and what the Tor client is used for (web or mixed workloads). Although the threat faced by clients used for non-web purposes is slightly lower, we find that it is growing at a faster rate than web-only clients. This is due to the small number of new non-Web supporting exit-relays being added to the Tor network.

Figure 3.9: The growth of the Tor network in terms of capacity, number of relays, and number of ASes.

Investigating further into the reason for the growth of the threat from AS-level adversaries in spite of the massive growth of the Tor network, we find that while the network has grown, the diversity of the ASes in the network has not increased. This is illustrated in Figure 3.9. Here, we see that while the number of relays in the network has grown to nearly 250% and the capacity of the network has grown to over 3000% of their 2010 values, the number of ASes in the network has lagged behind (growing to only 160% of its 2010 value).

Take-away: The Tor network faces a fundamental problem when dealing with AS-level attackers: the lack of AS-level diversity in the network. In the absence of a specific client-based solution for constructing AS-aware circuits, the threat from AS-level attackers is only expected to increase.

E5: Measuring the impact of asymmetric route exploiting adversaries. Recent work by Sun et al. [21] demonstrated, via high accuracy AS-level correlation attacks on the Tor network, that the threat from AS-level attackers was higher than previously anticipated. This is because of two factors: Adversaries can (1) exploit the asymmetry of routing on the Internet – i.e., exploit the fact that their presence on the forward- or reverse-paths at either end of the network is sufficient to launch an attack and (2) perform manipulation of routes via BGP hijacks and interceptions to place themselves on targeted paths. In this experiment, we consider the impact of adversaries on asymmetric routes. To do so, we repeat experiment E1, this time only considering an attacker that can exploit only forward paths – i.e., we say that a circuit is vulnerable to de-anonymization if there is some AS \( A \) such that: \( A \in \{ P_{SRC \to EN} \cap P_{EX \to DST} \} \). Figure 3.10 compares the fraction of websites marked as vulnerable against a forward-path exploiting (symmetric) adversary model with our (asymmetric) adversary model. We find that operating under the assumption of symmetric routing (i.e., considering only forward-path exploiting adversaries) results in threat under-estimation, with circuits to 17% of all websites identified as safe when they were in fact vulnerable.
3.3. MEASURING ADVERSARY PRESENCE

![Graph](image-url)  
**Figure 3.10:** Network-level traffic correlation attacks: Impact of asymmetric routes. Fraction of websites using vulnerable circuits against a symmetric and asymmetric adversary [E5].

**E6: Measuring the impact of BGP insecurity exploiting adversaries.** The potential for BGP hijacks and interceptions to compromise Tor traffic was highlighted by Sun et al. [21]. In this experiment, we measure how vulnerable Tor relays are to BGP hijacks and interceptions by sets of malicious ASes. We consider 10K pairs of (source, entry) ASes and 10K pairs of (exit, destination) ASes. The source ASes were randomly selected from the 1000 popular ASes (100 in each of ten countries) used in experiments E2-E4 while the entry and exit ASes were selected from the set of all Tor entry and exit relays, respectively. Destination ASes were randomly chosen from the set of all destination ASes seen in experiment E1 (web workload in 10 countries). For our adversary (i.e., ASes attempting to launch hijack and interception attacks), we selected the 16 malicious ASes identified in previous work [94] as popular ASes for hosting illegal content, botnet C&C servers, and other malicious resources. For each pair of ASes we use heuristics from Goldberg et al. [95] to identify malicious ASes capable of hijacking or intercepting traffic between the pair.

We first characterize the ability of the malicious ASes to hijack traffic for a chosen path. Figure 3.12 demonstrates the hijack and interception success rates of each of the 16 ASes considered in the experiment. We see that two ASes – ASN 9002: RETN (UA), ASN 29131: RapidSwitch (GB) – achieve high hijack and interception success rates of nearly 50%. The case of ASN 9002 can be explained by its high customer cone size (3271 customer ASes). On the other hand ASN 29131 is a smaller AS with only one customer AS, however, it peers with seven other large ASes having an AS rank under 1K (based on customer cone sizes). Figure 3.11 shows the fraction of hijack and interception attempts that were successful against the relays in ascending order. Each one of the relays we consider is susceptible to at least 20% of hijacks and 12% of interception attempts.

**E7: Measuring fraction of attacker-free paths.** We try to answer the question: “Is it possible to defend against network-level correlation attacks?” through this experiment. To do this, we measure the number of attacker-free entry- and exit-relay combinations available to the Tor client during relay selection. The intuition is: If a majority of our connections have no safe relay combination available to them (when considering all possible entry- and exit-relay combinations), no relay selection strategy can help mitigate the threat from AS-level correlation. On the other
hand, if there are safe alternatives available, it is possible to build a client that identifies and selects these combinations.

In this experiment, we replace our Ukraine (UA) VPN with a VPN from Iran (IR). This VPN was briefly available for a period of two weeks in 2015. We selected all the source and destination AS combinations observed in experiment E3 for analysis in this experiment (over 240K combinations). For each combination, we considered all entry- and exit-relay combinations that were available to the client handling the connection. Each generated (source AS, entry AS, exit AS, destination AS) combination was checked for safety. We see in Figure 3.13 the cumulative distribution function of the fraction of attacker free entry-exit pairs for each source-destination pair. Figure 3.13(a) shows this distribution for the five most vulnerable countries and 3.13(b) shows the distribution for the remaining countries.

China (CN) and Iran (IR) stand out as the most interesting cases. First, we see that 8% of all source-destination pairs have less than 10% of their entry-exit options being safe. Next, we also notice that there are no known attackers present on 18% of all source-destination pairs. This appears to indicate that the threat of de-anonymization is not uniform even within a country, with
3.3. MEASURING ADVERSARY PRESENCE

certain client locations being much safer than others. In order to understand which set of websites are more vulnerable in each of the countries, in Figure 3.14 we show the percentage of source-destination pairs having fewer than 5% safe circuit options for each set of websites. We find that in all cases, the Alexa top 100 local websites have fewer safe circuit options. This can be explained by the fact that locally popular websites are likely to be hosted within a regional AS.

![Graph showing distribution of attacker-free circuits for 100 source ASes](image)

(a) Most vulnerable countries (all websites): BR, CN, IR, RU, US
(b) Least vulnerable countries (all websites): DE, ES, FR, GB, IT

![Graph showing percentage of connections with less than 5% safe circuits](image)

Figure 3.13: Network-level correlation attacks: Availability of attacker-free options. Distribution of the fraction of attacker-free circuits for 100 source ASes with a web workload. More skewed to the right indicates the availability of more safe circuits [E7].

![Bar chart showing percentage of (source, destination) pairs](image)

Figure 3.14: (Logscale) Network-level correlation attacks: Fraction of connections with less than 5% safe circuits. Percentage of (source, destination) pairs having fewer than 5% attacker-free (entry, exit) options in each country [E7].

Through this experiment, we find that there are safe entry- and exit-relay combinations available during circuit construction, but these are not selected by the vanilla Tor client. This finding indicates that an AS-aware Tor client will be able to avoid AS-level traffic correlation in almost all cases.

**E8: Understanding the impact of number of guards.** In this experiment we consider the effect of the number of guards on the vulnerability of Tor clients to AS-level asymmetric correlation.
attacks. We selected the source, destination, and exit-relay AS triplets generated in experiment \textbf{E3} along with 60 unique guard-sets (20 each for 3 guards, 2 guards, and 1 guard) in an identical manner to the vanilla Tor client. Each resulting (source AS, entry AS, exit AS, destination AS) combination was checked for the presence of an AS-level adversary.

<table>
<thead>
<tr>
<th>Fraction of (source, destination) pairs</th>
<th>Fraction of attacker-free (entry, exit) pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 Guard</td>
</tr>
<tr>
<td></td>
<td>2 Guards</td>
</tr>
<tr>
<td></td>
<td>3 Guards</td>
</tr>
</tbody>
</table>

Figure 3.15: Network-level correlation attacks: Impact of number of guards on vulnerability. Distribution of the fraction of attacker-free (entry, exit) pairs for vanilla Tor configured to use 3, 2, and 1 guard(s) [E8].

Figure 3.15 illustrates the effect of reducing the size of the guard-set on the fraction of AS-level attacker-free-paths available to the Tor client. While it is known that a smaller number of guards provides better security against relay-level attackers in the long-term [19], we see from the results of this experiment that the effect is the opposite against network-level adversaries – i.e., as the size of the guard-set decreases, Tor is more likely to select a circuit vulnerable to network-level asymmetric correlation attacks due to the reduced number of available safe paths. In particular, when only 1 guard is used, over 15% of the (source, destination) pairs in our experiment have no safe-options, whereas the difference in security provided by two or three guards was marginal. This experiment demonstrates one of the conflicts between Tor clients geared for defending against relay-level attackers and those geared for defending against network-level attackers.

### 3.4 The Cipollino Tor Client

Motivated by the observation that vanilla Tor very often selects entry-exit pairs that may be subject to AS-level correlation attacks, we design a relay selection algorithm to mitigate the opportunities for such attackers. We design our relay selection system, Cipollino, based on the idea of probabilistic relay selection. This works by having the Tor client generate a probability distribution that minimizes the chance of attack over all possible entry- and exit- relay selection choices, and selecting an entry- and exit-relay based on this distribution. The advantage of such a relay selection approach is that even if the client has no safe options, relay-selection can be engineered to minimize the amount of information gained by the adversary over some period of time. Further, it allows clients to select relays in a way such that no set of relays in the Tor ecosystem is overloaded, even
3.4. THE CIPOLLINO TOR CLIENT

if every client uses the same relay-selection strategy. The Cipollino Tor client is designed with the following goals in mind:

- **Provide attacker free circuits for connections when possible.** Cipollino uses circuits that avoid active and passive network-level adversaries whenever they are available.

- **Minimize performance impact.** One of the primary barriers to the adoption of AS-aware Tor clients (e.g., Astoria [22] and LASTor [79]) has been their unsatisfactory performance – e.g., the median page-load cost (in seconds) is 2.3 seconds per page for Astoria. Cipollino counters these performance costs via a strategy of pre-emptive circuit construction.

- **Consider relay-level attacks.** A cost of using AS-aware Tor clients is the increase in vulnerability to relay-level attackers – e.g., constructing one circuit for each incoming request results in a large number of Tor relays being used by a single client. This increases the probability of the client being subject to relay-level attacks by a malicious relay. The Cipollino client circumvents this problem by employing a strategy of aggressive circuit re-use.

- **Perform load-balancing.** All circuits constructed by the Cipollino client explicitly take into account the bandwidth capacities of relays in the Tor network before including them in a circuit. As a result, no single set of relays are overloaded, even when a large number of users use the Cipollino client.

- **Consider the worst-case.** Cipollino employs a probabilistic circuit construction method to ensure that even in the case where there are is no safe circuit that can be constructed to a particular destination, the ability of a single attacker de-anonymizing a large number of Cipollino constructed circuits is minimized.

We now describe how the Cipollino Tor client achieves each of these goals.

3.4.1 Providing attacker free circuits

Cipollino considers an adversary model that includes the possibility of ASes exploiting (1) asymmetric routes and (2) BGP insecurities. To explain how we deal with such adversaries, we describe how Cipollino verifies the safety of a given circuit below.

- **Mapping destination IP addresses to ASNs and prefixes:** Given a circuit and a destination IP address, the Cipollino client first uses an up-to-date offline IP to ASN database (based off of BGP announcements) to obtain the AS numbers associated with the network of the client, entry-relay, exit-relay, and requested destination IP. This database (sourced and updated by CAIDA) is supplied and updated by the PathCache daily updates.

Following this, Cipollino generates two pairs of ASes and two pairs of prefixes – \((AS_{EN}, AS_{SRC}), (AS_{EX}, AS_{DST}), (Pre_{EN}, Pre_{SRC}), \) and \((Pre_{EX}, Pre_{DST})\).
• **BGP anomaly detection:** In order to detect hijacks and interceptions in near-real-time, Cipollino receives hourly (customizable in the client configuration) feeds from BGPStream [96] of current BGP routing anomalies. In particular, BGPStream produces a live stream of ongoing Multiple Origin AS (MOAS) anomalies. MOAS anomalies, which occur when a prefix is being announced by multiple origin ASes. We use MOAS as an indicator of potentially anomalous routing behavior as a proof of concept. Beyond the scope of the work in this dissertation, we are working to develop more accurate detection methods for hijacks and interceptions which could be incorporated into Cipollino [97]. This feed of ASes is used to identify ASes that are likely to be hijacking or intercepting traffic to any of the prefixes in the previously generated pairs \((\text{Pre}_\text{EN}, \text{Pre}_\text{SRC})\) and \((\text{Pre}_\text{EX}, \text{Pre}_\text{DST})\).

Any AS \(X\) that is suspected to be hijacking or intercepting traffic to the prefix associated with the entry-relay is added to the set \(H_{\text{EN}}\). Similarly, the sets \(H_{\text{EX}}, H_{\text{SRC}},\) and \(H_{\text{DST}}\) are populated.

• **Path prediction:** The Cipollino client uses the locally stored PathCache destination based graphs to obtain the set of ASes on the \(\text{SRC} \leftrightarrow \text{EN}\) and \(\text{EX} \leftrightarrow \text{DST}\) paths. Additionally, the ASes occurring on the paths between \(H_{\text{EN}} \leftarrow \text{SRC}\) and \(\text{EN} \leftarrow H_{\text{EN}}\) are added to \(\text{SRC} \leftrightarrow \text{EN}\). This accounts for all ASes that are able to view traffic characteristics in the event of a successful interception (and hijack) of traffic to \(\text{EN}\). The same process is repeated for \(H_{\text{EX}}, H_{\text{SRC}},\) and \(H_{\text{DST}}\).

• **Circuit safety marking:** After the paths are computed, a circuit is marked as safe iff the sets \(\text{SRC} \leftrightarrow \text{EN}\) and \(\text{EX} \leftrightarrow \text{DST}\) have no intersection.

The circuit safety verification procedure shows that Cipollino does not mark a circuit as safe to serve a given destination unless there are no ASes that are in a position to view traffic characteristics at either end of a circuit, after accounting for route asymmetry and potential hijacks.

Cipollino reduces dependence on algorithmic simulators by using PathCache [40] – a system that aggregates existing data and control plane measurements to predict paths. It falls back to simulations only when a path query cannot be answered using measurement data. Repeated querying of the PathCache server every time a circuit needs to be built is (1) time consuming and (2) reveals destinations of interest to a third party (e.g., PathCache server). To avoid this, the Cipollino client subscribes to daily updates of the routing graphs maintained by the PathCache server and locally computes paths between ASes. This is beneficial for two reasons:

1. **Offline verification of paths:** Since the meta-data for each edge in the routing graphs maintained by PathCache includes information regarding the source of the edge (i.e., to indicate the edge was observed in a traceroute from the RIPE Atlas network, control-plane data from RouteViews, etc.) and the measurement ID corresponding to the source. This is useful for the client to verify the authenticity of of a random subset of the daily updated paths supplied by the Cipollino aggregator.
2. **Low communication overhead:** The routing graph updates are between 5-15 MB/day. This is feasible for clients in most settings. Additionally, it allows clients to identify safe circuits even if the PathCache server is not immediately reachable.

We compare the security of the circuits constructed by Cipollino, Astoria, and the vanilla Tor client operating with the web workload in each of ten countries (same settings as experiment E1). The results are shown in Figure 3.16. We see that the Cipollino client circuits are safer against AS-level traffic-correlation adversaries. Only 1.4% of all web-pages loaded by the Cipollino client utilized a vulnerable circuit, compared to 11% and 57% for the Astoria and vanilla Tor clients, respectively.

![Figure 3.16: Security evaluation: Cipollino vs. Astoria vs. vanilla Tor.](image)

### 3.4.2 Reducing performance overhead

One of the primary barriers to the adoption of AS-aware Tor clients (e.g., Astoria [22] and LASTor [79]) has been their unsatisfactory performance. Cipollino counters these performance costs via a strategy of pre-emptive circuit construction.

Cipollino employs a circuit pre-building strategy similar to the vanilla Tor client – *i.e.*, Cipollino pre-emptively constructs a fixed (and configurable) number of circuits. This approach is motivated by the following two observations:

1. The results of experiment E7 show that for over 50% of all client locations and destination ASes considered, at least 50% of all possible entry- and exit-relay combinations were safe from correlation attacks by AS-level adversaries. Therefore, by pre-building a number of circuits, we are very likely to find at least one safe circuit for a given destination AS.

2. Constructing a new circuit is significantly more expensive than verifying the safety of an existing circuit – *i.e.*, due to the need for estimating the paths between all possible (source, entry-relay) and (exit-relay, destination) pairs. Therefore, by pre-emptively constructing circuits, Cipollino reduces the need to construct on-demand destination-aware circuits.
Figure 3.17 illustrates that pre-building circuits results in the need for constructing fewer on-demand and destination-aware circuits. In this experiment, 1,000 Cipollino clients were simulated (with locations in the 100 most populous ASes in each of ten countries) and issued connection requests for destinations associated with the web workload. Here we see that 50% of the clients were able avoid on-demand circuit construction for at-least 86% of the connection requests, when just four circuits were prebuilt.

![Graph showing pre-building circuits results in fewer on-demand and destination-aware circuits](graph1.png)

Figure 3.17: Cipollino performance tuning: Impact of pre-emptive circuit construction on safe circuit availability. Distribution of the fraction of connection requests that were able to find a safe and usable circuit from 4, 16, and 64 circuits pre-built be the Cipollino client.

Reusing circuits, when possible, also improves the performance of Cipollino compared to other AS-aware Tor clients. Figure 3.18 shows the elapsed time between the arrival of a connection request and the allocation of a circuit to satisfy the request. As expected, since the vanilla Tor client always uses an existing circuit, it is faster than Astoria and Cipollino, requiring under .1 seconds to allocate a circuit to over 99% of incoming connection requests. Within the same time constraints we see that the Cipollino Tor client is able to satisfy 60% of its requests, while the Astoria client can only satisfy 21%.

![Graph showing circuit allocation times](graph2.png)

Figure 3.18: Cipollino performance tuning: Impact of pre-emptive circuit construction on circuit allocation times. Distribution of circuit allocation times under different pre-emptive circuit construction configurations.
3.4. THE CIPOLLINO TOR CLIENT

To give a complete picture of the performance of the Cipollino client we consider the time required to load a complete web-page (including third-party content). Figure 3.19 shows the cumulative distribution of page-load times of 2000 web-pages in ten client locations for the Cipollino, Astoria, and Tor clients. We find that the time required for loading pages using the Cipollino and Tor client are quite closely matched with the median page-load time differing by only 1.6 seconds, while the Astoria Tor client is nearly 7 seconds slower. This validates our pre-emptive circuit construction approach.

![Figure 3.19: Performance evaluation: Cipollino vs. Astoria vs. vanilla Tor.](image)

3.4.3 Considering relay-level attacks

We argue that a client which utilizes a smaller number of relays to serve connection requests, over a period of time, is less likely to encounter a malicious relay in the Tor network. Thus, a Tor client that uses a smaller number of relays is more secure against adversarial relays. We observe that many proposed defenses [69, 77, 79, 22] do not consider the impact of AS-aware relay selection on the security of the client against relay-level adversaries. This is problematic because many AS-aware clients build circuits on a per-destination basis, as opposed to reusing a smaller set of existing circuits. This results in them leveraging a large set of relays over time.

![Figure 3.20: Cipollino security tuning: Impact of pre-emptive circuit construction on number of unique relays utilized in Cipollino circuits.](image)
The pre-emptive circuit construction strategy employed by Cipollino has a side-effect of reducing the number of relays utilized by its circuits. To understand how circuit pre-building affects the number of relays used by Cipollino, we execute the web workload on the Cipollino client configured to pre-build and always maintain 4, 16, and 64 live and usable circuits. Figure 3.20 compares the number of unique relays used in each setting with the vanilla Tor client and Astoria. When Cipollino is configured to only pre-build and maintain 4 active circuits, it utilizes 786 relays (compared to the 623 relays used by Tor). This is lower than Astoria (3104 relays).

3.4.4 Load-balancing across relays

Tor is a system run using community resources contributed by volunteers. Therefore, load-balancing connections across these resources is important to ensure that they are used efficiently and no single relay or set of relays is overloaded. Load balancing is explicitly performed in two cases: (1) when constructing and replenishing Cipollino’s reserve of pre-built circuits and (2) when there are multiple safe circuits available for a connection request.

In the first case, Cipollino exactly mimics the load-balancing approach utilized by the vanilla Tor client – i.e., relays are selected in a circuit with probability proportional to their bandwidth capacity. The second case, however, is more nuanced. When there are multiple safe entry- and exit-relay options – \((e_n, x_1), \ldots, (e_n, x_n)\) – Cipollino selects the \(i^{th}\) entry and exit-relay combination with probability \(Pr_i\), where:

\[
Pr_i = \frac{BW_{en_i} \times BW_{ex_i}}{\sum_{j=1}^{n} BW_{en_j} \times BW_{ex_j}}
\]

Here \(BW_{en_j}\) and \(BW_{ex_j}\) are the advertised bandwidths of the entry- and exit-relay associated with the \(j^{th}\) safe relay combination. This weighting of combinations works to ensure that each entry- and exit-relay is selected with the probability proportional to its advertised bandwidth (when only considering safe relay options).

![Figure 3.21](image)

Figure 3.21: Load-balancing: Cipollino vs. Astoria vs. vanilla Tor. Distributions of the bandwidths of the relays selected by Cipollino, Astoria, and the vanilla Tor client.
3.4. THE CIPOLLINO TOR CLIENT

Astoria, and Cipollino. We find that they are all able to effectively ensure that relays do not get overloaded. Further, Cipollino does not perform any worse than Astoria, despite its reuse of existing safe circuits.

3.4.5 Considering the worst-case

Although there is a safe entry- and exit- relay combination for most connection requests, we develop our relay selection to be robust even if this is not the case. To minimize the risk of correlation attacks, we define a linear program which generates a probability for each relay selection with the objective to minimize the maximum probability of a circuit encountering the attacker. Recall that in our adversary model, we consider a long-lived adversary and that minimizing the probability of an attacker may also be seen as minimizing the number of circuits the adversary is able to observe over a long period of time and numerous circuit construction cycles.

Figure 3.22: Cipollino: Optimizing relay selection in the worst-case. A simplified example using only unidirectional paths and only entry-relay selection.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Uniform</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS1</td>
<td>1/3</td>
<td>1/4</td>
</tr>
<tr>
<td>AS2</td>
<td>1/3</td>
<td>1/4</td>
</tr>
<tr>
<td>AS3</td>
<td>1/3</td>
<td>1/2</td>
</tr>
</tbody>
</table>

Figure 3.22 shows an example of worst-case relay selection to give intuition about how the LP minimizes the risk from the attacker. In this example, we consider unidirectional paths and only entry-relay selection for clarity. In the figure, if the source were to choose uniformly at random across the three entry-relays, there is a 2/3 chance that AS1 will be able to observe traffic and only a 1/3 chance that AS2 will. In this case, the optimal selection is intuitive, that the source should choose entry-relays 1 and 2 with probability 1/4 each and entry-relay 3 with probability 1/2. This lowers the probability that AS1 can observe a circuit from 2/3 to 1/2. This probability of the most likely adversary is the quantity that our LP minimizes.

We use the following notation:

- Let $ADV_{i,j}$ be the set of attackers on the circuit using entry-relay $i$ and exit-relay $j$ to destination $dest$ – i.e., $\forall A \in ADV_{i,j} : A \in \{p_{src \leftrightarrow entry_i} \cap p_{exit_j \leftrightarrow dest}\}$.

- Let $X_{i,j,A}$ be an indicator random variable for attacker $A$ on the circuit using entry-relay $i$ and exit-relay $j$ – i.e., $X_{i,j,A} = 1 \iff A \in ADV_{i,j}$, and 0 otherwise.

- Let $P_{i,j}$ be the probability that a client builds a circuit using entry-relay $entry_i$ and exit-relay $exit_j$. 
The following linear program is used to minimize the probability of the most likely attacker (i.e., the number of circuits visible to the attacker).

\[
\begin{align*}
\text{minimize } & \quad z \\
\text{subject to } & \quad z \geq \sum_{i,j} (P_{i,j} X_{i,j,A}) \quad \forall A \in ADV_{i,j} \\
& \quad P_{i,j} \in [0, 1], \forall i, \forall j; \sum_{i,j} P_{i,j} = 1 
\end{align*}
(3.3)
\]

Essentially, given information about the presence of attackers (network-level or state-level) for each \(p_{\text{source} \rightarrow i}\) and \(p_{j \rightarrow \text{dest}}\) path, the linear program seeks to find the probability distribution \((P_{i,j})\) over available choices of entry- and exit-relays, for which the expected number of circuits visible to each attacker is minimized. Entry- and exit-relays are chosen according to this distribution (defined as \(D_{lp}\)) during circuit construction.

### 3.4.6 Putting it all together: The Cipollino client architecture

Cipollino consists of three main components: (1) an AS-level path aggregation toolkit (PathCache), (2) a circuit allocator, and (3) a circuit builder. The interaction between each of these components is illustrated in Figure 3.23.

![Figure 3.23: Cipollino: System architecture.](image)

The Cipollino client maintains a compact local repository of destination-based routing graphs. These are updated by the PathCache servers on a daily (or, configurable) basis. The PathCache path-stitching algorithms are used on these graphs to identify ASes that are in a position to observe
traffic flowing between a given source and destination AS.

When the Cipollino client receives a request for a connection to a destination IP and port, the circuit allocator uses the PathCache stitching algorithms and graphs to identify if there are any pre-built circuits that are not vulnerable to traffic correlation attacks by ASes. If exactly one of the safe circuits is able to serve the requested IP and port of the destination, then the circuit is used to satisfy the connection request. If there are multiple such circuits, then one of them is chosen in accordance with our load-balancing scheme described in the previous section.

In the event that none of the pre-built circuits is able to satisfy the connection, the circuit builder constructs a circuit specifically for the requested connection. The constructed circuit performs also relay selection in a way that achieves load-balancing across all relays in the Tor network. Additionally, the circuit builder also handles the worst-case scenario – when there are no safe circuits that may be built. In this case, the circuit builder borrows the linear program proposed by the Astoria Tor client to ensure that no single adversary is able to deanonymize a large number of circuits.

3.5 Conclusions

In this chapter we analyzed the threat faced by Tor clients from AS-level adversaries from a current and historical perspective. We found that the current threat is high, with around 30% of all Tor circuits created in our experiments remaining vulnerable to deanonymization by AS-level correlation attacks, regardless of whether the Tor client is used for web browsing or other applications. Further, our historical analysis points to a fundamental problem with the Tor network the lack of growth of AS-level diversity. Without specific efforts from the Tor project to increase diversity of relays or incorporate AS-awareness in the Tor client, our study shows that the threat is bound to increase.

We show how high under-estimation of the threat from AS-level adversaries, or increased vulnerability to active (AS-level) and passive (relay-level) adversaries, or poor performance characteristics is possible when an AS-aware Tor client ignores realities such as the possibility of asymmetric and BGP exploiting adversaries. We find that our AS-aware Tor client Cipollino, designed specifically to address these pitfalls improves the current state-of-the-art by achieving better security against network-level adversaries. Specifically, by using a data- and control-plane measurement infrastructure whenever possible, Cipollino reduces the fraction of vulnerable webpage loads from 57% (vanilla Tor) and 11% (Astoria) to 1.4%. Additionally, by incorporating the concept of circuit pre-building and circuit re-use, the Cipollino client significantly reduces the threat faced from malicious relays. As a consequence of circuit pre-building and re-use, the Cipollino client is also able achieve performance characteristics comparable with the vanilla Tor client. Our work highlights the importance of applying current models and data from network measurements to inform relay selection so as to protect against timing attacks.
Chapter 4

Circumventing Network-level Blocking

Contents

4.1 Introduction ................................................. 68
  4.1.1 The promise of video games ......................... 69
4.2 Adversary and Threat Model ................................. 71
  4.2.1 Network traffic attacks .......................... 72
  4.2.2 Application layer attacks ..................... 72
  4.2.3 Censor limitations ............................ 72
4.3 The Castle Circumvention Scheme ......................... 73
  4.3.1 Real-time strategy games .......................... 73
  4.3.2 Building game-based covert channels .......... 75
  4.3.3 Encoding data into game commands .......... 76
  4.3.4 Sending covert data ............................ 77
  4.3.5 Receiving covert data ........................... 78
  4.3.6 Bootstrapping Castle communication ............ 79
  4.3.7 Castle prototype implementation ............. 79
4.4 Evaluation .............................................. 81
  4.4.1 Security Evaluation ............................... 81
  4.4.2 Performance Evaluation ............................. 86
  4.4.3 Castle system requirements ........................ 89
4.5 Extensibility of Castle .................................... 89
4.6 Castle and Other Look-like-something Circumvention Tools . 90
4.7 Conclusions .................................................. 91
In this chapter, we explore video games as a new avenue for covert channels. Two features make video games attractive for use as a cover protocol in censorship circumvention tools: First, games within a genre share many common features. Second, there are many different games, each with their own protocols and server infrastructures. These features allow circumvention tool developers to build a single framework that can be adapted to work with many different games within a genre; therefore allowing quick response to censor created blockades. In addition, censored users can diversify their covert communications across many different games, making it difficult for a censor to respond by simply blocking a single covert channel. The results presented in this chapter have previously appeared as part of the following article: [33].

4.1 Introduction

The Internet has become a critical communication infrastructure for citizens to obtain accurate information, organize political actions [98], and express dissatisfaction with their governments [99]. This fact has not gone unnoticed, with governments clamping down on this medium via censorship [100, 101, 102], surveillance [3] and even large-scale Internet take downs [103, 104, 105]. The situation is only getting worse, with Freedom House reporting 36 of the 65 countries they survey experiencing decreasing levels of Internet freedom between 2013 and 2014 [90].

Researchers have responded by proposing several look-like-something censorship circumvention tools. These tools aim to disguise covert traffic as another (benign) protocol to evade detection by censors. This can take two forms: either mimicking the cover protocol using an independent implementation, as in SkypeMorph [34] and StegoTorus [106], or encoding data for transmission via an off-the-shelf implementation of the cover protocol, as in FreeWave [35].

This has created an arms race between censors and circumvention tool developers. For example, Tor’s introduction of “pluggable transports”, i.e., plug-ins that embed Tor traffic in a cover protocol to counter censors that block Tor [107]. Censors have already begun blocking some of these transports [108], and some censors have gone so far as to block entire content-distribution networks that are used by some circumvention systems [109].

Furthermore, recent work has shown that care must be taken when designing and implementing a look-like-something covert channel. For example, Houmansadr et al. showed that, when a covert channel re-implements its cover protocol, the copy is unlikely to be a perfect mimic of the original protocol, and a censor can use the differences to recognize when a client is using the covert channel [110]. Worse yet, Geddes et al. demonstrate that even running the cover application is not enough to avoid detection by censors [111] – i.e., approaches like FreeWave may be detected via architectural, channel, and content mismatches between the application’s regular behavior and its behavior when being used as a covert channel.
4.1. INTRODUCTION

In light of this state of affairs, our work argues that video games have several features that make them an attractive target for covert channel development.

There are many games available, enabling developers to create a diverse set of circumvention tools. The number of real-time strategy games has grown rapidly in the last few years. This growth has been driven in part by the democratization of game publishing, as embodied in game distribution platforms such as Steam [112] - e.g., Figure 4.1 shows the total number of real-time strategy video games that have been released since 2010 on the Steam platform. Further, each game uses its own network protocol and infrastructure, so the censor cannot simply block all games using a single technique. Censorship circumvention developers can use this large body of games to adapt and evade a censor’s attempt to block any particular game.

Video games share common elements, making it possible to use a single framework across many games. For example, most Real-Time Strategy (RTS) games have the notions of buildings, units, and rally points, and censorship circumvention tools that encode information by interacting with these objects can be easily ported from one RTS game to another. Many games also feature replay logs and similar user interfaces, enabling covert channel frameworks that are only loosely coupled to the internals of any particular game.

Game-based circumvention tools can re-use off-the-shelf game implementations. Since games have features that make it relatively easy to automate interaction with the game, circumvention tool developers do not need to re-implement the game (or its network protocol), ensuring that the circumvention tool can leverage the existing implementation of the game. This prevents attacks that can distinguish between the original implementation and the cover-protocol implementation of an application or protocol [110].

Game-based circumvention tools avoid previously encountered pitfalls. Games in select genres often support both peer-to-peer and server-based gaming sessions (e.g., real-time strategy games), so they can adapt to whichever is better for the circumvention tool. This allows architectural matching as described by Geddes et al. [111]. Games must maintain synchronized state, so they are loss sensitive, avoiding the channel mismatch between multimedia and Web/textual covert content identified by Geddes et al. [111]. Finally, games are reasonably able to avoid content
mismatches by due to the large amount of diversity in typical content characteristics.

**Games often have built-in security features that can support secure covert channels.** It is considered good practice for games support encryption and authentication in order to prevent cheating [113] – e.g., the Microsoft DirectX networking API [114] and the Steam peer-to-peer networking API [115] which are commonly used by game developers include support for SSL sockets. Additionally, some games also support password-protected sessions, which can prevent application-level attacks in which the censor attempts to identify covert channels by joining the game.

**Games have the potential to reverse the resource imbalance in the arms race between censors and developers.** By lowering the development cost of creating new covert channels, video games can create an asymmetry that circumventors can use to win the arms race against censors. Censors can respond to look-like-something circumvention tools by blocking the cover protocol entirely or attempting to distinguish legitimate uses of the protocol from uses by the covert channel. If developing such mechanisms is time consuming for the censor, but circumvention tool developers can quickly construct new tools, there will almost always be effective circumvention tools available for end users.

In spite of the above benefits, we must answer several questions to understand the feasibility of using video games for covert channels:

- **Security:** Can we encode data in the video game so that the censor cannot distinguish regular game play from covert channel sessions?

- **Extensibility:** Can we build a framework that can be quickly adapted to new games?

- **Performance:** Can video games support good covert channel bandwidth?

To answer these questions, we have built Castle, a prototype video game-based covert-channel framework. Castle encodes data as player actions in an RTS game. Castle uses desktop-automation software to execute these actions in the game. The video game software transmits these moves to the other players in the same gaming session, who then decode the message and send replies in the same way.

**Security.** Castle’s design makes it resilient to several classes of attacks. Since Castle uses the underlying game to transmit data, an attacker cannot use simple IP- or port-based blocking to block Castle without blocking the game entirely. When used with games that encrypt and authenticate their traffic, an attacker cannot use deep packet inspection to distinguish Castle traffic from regular game traffic. Encryption and authentication also preclude simple packet injection or manipulation attacks. Since games use network communication to synchronize their state, they are loss sensitive, unlike some VoIP protocols. Thus Castle cannot be distinguished from regular gaming sessions through selective packet delay or dropping attacks. Finally, when used with password-protected gaming sessions, Castle is immune to application-level attacks, such as the censor attempting to join the same gaming session to observe the player’s in-game actions.
We evaluate Castle’s security against statistical traffic-analysis attacks by applying several previously published classifiers – *i.e.*, the Liberatore [53], Herrmann [55], and Shmatikov [116] classifiers. We find that packet sizes and inter-packet times of Castle’s traffic deviate from those of regular human-driven game play by the same amount that different human player’s traffic differ from each other.

**Extensibility.** Castle can be easily adapted to new RTS games. Our current prototype supports three such games: “0-A.D.” [117] and two extremely popular (over 8.5 million copies sold) closed-source games from different development studios that we refer to as “Aeons” and “Conquerors”. It took a single undergrad less than six hours to port Castle from 0-A.D. to each game.

Castle is easy to port to new RTS games for two reasons. First, Castle uses only features that are nearly universal to RTS Games – *e.g.*, game-play characteristics and game replay features. Thus the high-level architecture and encoding scheme can be re-used across games. Second, Castle is only loosely coupled to game internals – requiring no access to the game source-code. For example, Castle uses desktop-automation software to execute game actions through the game’s standard graphical user interface. As a result, Castle does not need to understand the game’s network protocol or any other internals.

**Performance.** Castle offers good bandwidth for text-based communications. Our current prototype provides between 50 and 200 B/s of bandwidth, depending on configuration parameters. Castle has about 100x more bandwidth than other proposed game-based covert channels [118, 119, 120]. With some game-specific tuning, the Aeons version can deliver over 400 B/s. Even 50 B/s is sufficient for bootstrapping high bandwidth communication channels (*e.g.*, distributing Tor bridge IPs), text-based web articles, email, SMS messages, tweets, and other asynchronous communications which are widely used organizational tools among political activists. There are also several ways to potentially increase Castle’s bandwidth (see Section 4.4.2 for details).

Together, these results show that video games offer promise as a target for covert channel development and they may enable circumvention tool developers to gain the upper hand in the arms race against censors.

### 4.2 Adversary and Threat Model

In this chapter, we consider a network-level censor (*e.g.*, an ISP) able to (1) perform analysis over all traffic that it forwards from or to clients within its network and (2) perform manipulations (*e.g.*, dropping and injecting packets) of the network traffic via on-path or in-path middleboxes. In addition, the adversary may also take an active approach by probing and interacting with application endpoints.

---

1. Despite the similarity of their names and their common use of video games, Rook and Castle were developed independently and have quite different goals. See Section 4.6 for details.
4.2.1 Network traffic attacks

**Passive analysis.** We consider censors that are able to perform stateless and stateful passive analysis of traffic at line rate. In particular, the censor is able to perform the following passive analyses to detect the use of a circumvention tool:

- **IP and port filtering:** The censor can observe the IP addresses and port numbers of connections on their network (e.g., using tools like Netflow [121]).

- **Deep-packet inspection:** The censor may look for specific patterns in packet headers and payloads (e.g., payloads indicative of a specific game).

- **Flow-level analysis:** The censor may perform statistical analyses of flow-level characteristics such as inter-packet times and sizes) while maintaining a reasonable amount of state.

The first two of these capabilities mean that the ISP can detect flows related to the video game in general. For example, if the game uses a specific set of servers (IPs) or ports, these flows may be easily identified. Similarly, game-specific payloads can reveal game traffic to the ISP. The last property can reveal information about game behavior to the ISP. A circumvention system must avoid perturbing these features to remain undetected and unblocked.

**Active manipulations.** In order to detect and/or disrupt the use of censorship circumvention tools, censors may perform a variety of active manipulations on suspicious connections that transit its network. In particular, the censor may drop, insert, or delay packets. Additionally, they may also modify the packet contents and headers. The adversary may perform these manipulations to observe the behavior of flow endpoints to distinguish legitimate game traffic from the covert channel. They may also use these actions to block covert connections (e.g., sending TCP RST packets, or dropping traffic).

4.2.2 Application layer attacks

In the context of detecting *look-like-something* covert channels, censors may take additional actions outside the scope of standard active and passive analysis. Specifically, they may interact with the application that the covert channel aims to hide within. They may attempt to join game servers and observe games in progress (i.e., to identify who is playing with whom). Additionally, they may seek to observe properties of the games being played (e.g., map state, player move behaviors) or join and interact with game players.

4.2.3 Censor limitations

We impose limitations on the computational and storage capabilities of censors. While they have a large amount of computational resources, they are still unable to maintain a large amount of per-connection state for long durations or decrypt encrypted communication channels and guess high entropy passwords. We also assume that the censor does not have a back door into the game
or game servers. For example, we assume the censor is not able to break into the game servers (e.g., by exploiting a buffer overflow or other bug). We also assume that the operators of the game servers do not cooperate with the censor – i.e., they do not allow the censor to see other user’s private game state.

4.3 The Castle Circumvention Scheme

Castle aims to demonstrate that highly portable, secure, and low-bandwidth look-like-something defenses are possible via applications such as real-time strategy video games. In this section, we provide a background on the real-time strategy genre and highlight key properties of these games that enable Castle to create covert channels that generalize to a large number of games within the genre. Finally, we describe how Castle encodes, sends, and receives data.

4.3.1 Real-time strategy games

Real-time strategy games are a genre of video games that center around the idea of empire-building. Typically, the goal is for a player to assert control over enemy territory through a combination of military conquest and economic maneuvering. Below we highlight commands and features that are common to a large majority of real-time strategy games (Table 4.1) and are critical to the extensibility of Castle.

Units. Real-time strategy games allow players to create and train a large number of units (e.g., human characters, livestock, machinery). Units may perform many actions. For example, in 17 of the Top 20 best-selling real-time strategy games [122], a unit can be instructed to move to a location on the map by left-clicking it and then right clicking the destination location on the map.

Buildings. Players may construct a number of buildings over the course of a game. Buildings are required to train certain units and research new technologies – e.g., barracks are required to train infantry. In many (e.g., in 17 of the Top 20 best-selling) real-time strategy games, unit-producing buildings can be assigned a rally point – i.e., a location at which all units created by the building will assemble.

Maps and map editors. Real-time strategy games are set in a landscape covered by plains, forests, mountains, and/or oceans. Many (including 17 of the Top 20 best-selling) real-time strategy games allow users to create and use their own maps, or modify existing maps for use within the game. This is either from within the game, or via external mods.

Replay files. In newer games, players may be given the option to record all moves and commands issued by themselves and other players in the game. This is used to replay or watch previously played video games. When this option is enabled, the game writes, in real-time, all in-game commands to a replay log. While this replay log may be stored in a proprietary format, we found decoders to read these formats are available for 11 of the Top 20 best-selling real-time strategy games.

In addition to the above elements, the following networking and security properties are standard
CHAPTER 4. CIRCUMVENTING NETWORK-LEVEL BLOCKING

<table>
<thead>
<tr>
<th>Feature</th>
<th>Number of Games</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Commands</td>
<td>17</td>
</tr>
<tr>
<td>Map Editors</td>
<td>17</td>
</tr>
<tr>
<td>Replay Decoders</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 4.1: Real-time strategy game features used by Castle and the number of games in the Top 20 best-sellers of all-time that possess them [122].

Network communications. For scalability reasons, real-time strategy games do not broadcast state information to all players in the game. Instead, they pass commands issued by the players in fixed intervals (e.g., 100 ms). These commands are then simultaneously simulated in each game client. This allows clients to execute the game identically, while requiring little bandwidth [123]. As a consequence, any data encoded as an in-game command is received as such, by other players.

Additionally, while most real-time strategy games make use of UDP channels for command communication, reliable delivery mechanisms are implemented in the application layer. This makes many active traffic manipulation attacks described in previous work [111] ineffective.

In terms of network architecture, real-time strategy games may take two forms, with players joining a game hosted on a public game server (e.g., servers hosted by game publishers such as Microsoft, Blizzard, Electronic Arts, etc.), or connecting directly to each other in a peer-to-peer mode. Therefore, any covert channel system utilizing video games as a cover, can employ whichever is the dominant mode of operation and shift from one to the other if required, to evade a censor blockade.

Security considerations. In order to prevent cheating in the multi-player setting, it is considered good practice to implement encrypted and authenticated communication channels, in real-time strategy games [113]. Additional mechanisms such as verification of game-state consistency (between all clients in the game) [123] and password-protected multi-player game sessions [115] are also common.

These security mechanisms have several vital consequences in the context of using real-time strategy games as covert channels. First, since the game command channel is encrypted, passive adversaries are unable to view commands issued by players in a game by simply observing network traffic. Second, the presence of authenticated channels and game-state verification algorithms prevents active attackers from using falsified game packets to interact with, or observe other clients on the game servers. Finally, the presence of password-protected game instances prevents adversaries from joining multi-player games (to observe the in-game state and identify players).

Commonalities between real-time strategy games. Our design leverages the common command structure, map design capabilities, and tools for decoding saved games and replays generated by real-time strategy games. Table 4.1 shows the results of our survey of the prevalence of these
4.3. THE CASTLE CIRCUMVENTION SCHEME

Figure 4.2: Castle architecture: Overview of data flow for sending and receiving in Castle. Shaded components are implemented as part of Castle while the others use existing off-the-shelf software.

features in real-time strategy games. We find that 11 of the top 20 best-selling games of all-time include these features.

4.3.2 Building game-based covert channels

**Straw-man approach.** One may consider establishing covert channels via the in-game voice or text chat channels. However, this approach has several drawbacks.

First, previous work shows that encoded data is easily distinguishable from human audio communication [111, 110]. Furthermore, voice communication channels are fairly uncommon in the real-time strategy game genre. Second, while game data is encrypted, it is often the case that text communication channels are left unencrypted. Finally, while one may expect a fairly constant stream of human issued in-game commands in a real-time strategy game, it is rare to have long text or audio communication while playing the game. These factors allow covert channels built on these approaches to be either difficult to implement/extend, or to be easily detected by an adversary, or both.

**The Castle approach.** To create a covert channel mechanism that is extensible to a large number of games in the real-time strategy genre, Castle exploits two key properties.

- **Presence of common commands.** Real-time strategy games share a common set of commands. Specifically, the ability to select buildings and assign a location where units created in these building should assemble. This location is called a “rally point”. We denote the command of setting the rally point for units created in a given building by `SET-RALLY-POINT`. Games also provide the ability to move a selected unit to a given location (denoted by the `MOVE` command). Thus, any encoding that translates covert data into a combination of unit/building selections and these primitives will be extensible across games in the genre.

- **Access to replay logs.** Often, real-time strategy games provide a replay option which saves every players’ moves to disk (for later playback). Therefore, all in-game commands are written to disk where they can be read and decoded in real-time.
Castle consists of two main components to send and receive data. These are illustrated in Figure 4.2. Sending is done by encoding data into game commands and then executing them within the game using desktop automation. The receiving process monitors the log of game commands and decodes this list to retrieve data sent via the system.

4.3.3 Encoding data into game commands

Castle relies on the ability of the player to select units and buildings and set rally points to encode data. A naive encoding may consider selecting each unit and directing it to a different point on the game map to encode a few bytes of information per unit. However, in preliminary experiments, we observed that this approach resulted in a covert channel that could not match the properties of the original game traffic (moving O(100s) of units to distinct locations is not a usual action for players).

Encoding in Castle is accomplished, without inflating the amount of game data transferred, using the following scheme. First, the participants in Castle use (standard or Castle-specific) maps which contain either \( n \) immobilized units (e.g., units placed in unit sized islands, within walls, etc.) or \( n \) unit producing buildings (e.g., barracks, stable, etc.). The Castle sending process then encodes data by selecting a subset of these \( n \) units and executing either a MOVE command in the case of units or SET-RALLY-POINT in the case of buildings. While we discuss the encoding in the context of units and the MOVE command, Castle is easily implemented using either primitive.

Instead of using each of the \( n \) units to represent a single bit sequence, which would result in \( \log_2(n) \) bits of data transferred per command, we use a combinatorial scheme where we select \( k \) of the \( n \) units, to increase efficiency. Intuitively, the selection of \( k \) of \( n \) units results in \( \binom{n}{k} \) different values or \( \log_2 \binom{n}{k} \) bits that may be transferred per command. We use combinatorial number systems [124] to convert \( \log_2 \binom{n}{k} \) bits of data into a selection of \( k \) of the \( n \) units. In preliminary experiments, we found that the selection of a constant number of units per command resulted in traffic which was more uniform than regular game traffic. As a result, we adjusted our scheme to select between 0 and \( k \) units for encoding to increase variability of packet sizes. Section 4.4.1 provides a more in-depth view of how we evaluate our similarity to actual game traffic.

In addition to selecting the set of units, we can also select a location for all \( k \) selected units to move to. Note that since we select a single location for \( k \) units (instead of \( k \) distinct locations) this does not impact the data transfer size. Given a game map with \( m = x_{\text{max}} \times y_{\text{max}} \) potential locations we can additionally encode \( \log_2 m \) additional bits of data in a given turn.

Assuming a map with \( n \) units/buildings, a maximum of \( m = x_{\text{max}} \times y_{\text{max}} \) map locations, and a game which allows for a maximum of \( k \) units/buildings to be selected simultaneously, the game-independent encoding of covert data into a MOVE or SET-RALLY-POINT command is done as shown in Algorithm 4.7.

The combination of selecting between 0 and \( k \) units and setting the location to move to, results in an average of
Algorithm 4.7 Algorithm for encoding covert data into game commands

function \text{ENCODE}(data, k, n, m, x_{\text{max}}) \\
\hspace{1em} r \leftarrow \{1, \ldots, k\} \\
\hspace{1em} z_1 \leftarrow \text{READ}(data, \log_2 \binom{n}{r}) \\
\hspace{1em} \text{for } i = n \rightarrow 0 \text{ do} \\
\hspace{2em} \text{if } \binom{i}{r} \leq z_1 \text{ then} \\
\hspace{3em} z_1 \leftarrow z_1 - \binom{i}{r} \\
\hspace{3em} selected \leftarrow selected || i \\
\hspace{2em} \text{end if} \\
\hspace{1em} \text{end for} \\
\hspace{1em} z_2 \leftarrow \text{READ}(data, \log_2 m) \\
\hspace{1em} (x, y) \leftarrow (z_2 \mod x_{\text{max}}, \lfloor z_2/x_{\text{max}} \rfloor) \\
\hspace{1em} \text{return } \{\text{selected}, (x, y)\} \\
\end{function}

function \text{READ}(file, b) \\
\hspace{1em} \text{return } \text{next } b \text{ bits from } file \text{ in base 10.} \\
\end{function}

\[ \left( \sum_{i=1}^{k} \log_2 \binom{n}{r} \right) / k + \log_2 m \] bits transferred per command.

As mentioned earlier, one may achieve higher data-rates by always selecting \( k \) units, however, this causes identically sized commands and thus affects the packet size distribution.

### 4.3.4 Sending covert data

Once the covert data is encoded into in-game commands, the sending process must actually execute the commands in order to communicate them to the receiver. One way to do this is to modify the game AI to issue commands as dictated by our encoder. However, this is non-trivial since most games are closed-source and viewing/modifying game code is not always an option. Even when source code is available, the overhead of understanding the game code and modifying the AI presents a non-trivial hurdle. Given our vision of adaptability to the large number of available real-time strategy games, we leverage off-the-shelf desktop automation to execute the encoded game commands. This opens the door to extending our approach to a larger set of games than would otherwise be possible.

Since the map used in Castle is custom made, the starting location of all units is known in advance. Further, since units and buildings are immobile, Castle is aware of their location at all times. The location of units on the game map, along with the list of commands to be executed is sufficient for Castle to automatically generate a sequence of key-presses, and clicks to be made by the desktop automation tool. This sequence is then passed to the automation tool for execution.

We note that, certain automation tools allow keystrokes and clicks to be sent to windows that are not currently in focus. This ensures that Castle does not detract from the user experience by requiring the game window to be in focus during data transfer periods. Finally, since automation tools allow control over the speed of clicks and key-presses, Castle can be configured to either
mimic human input speeds (lower clicks/second) or maximize throughput (higher clicks/second). We investigate the trade-off between these two variables in Sections 4.4.1 and 4.4.2.

4.3.5 Receiving covert data

Since the receiving game client does not have the same in-game screen as the sending client (due to each client having their camera focused on different map locations), directly observing the commands made by the sending client via the screen output is prohibitively complex. Fortunately, most real-time strategy games maintain a real-time log of all commands issued in the game to enable replaying moves or saving game state. In Castle, the receiving process constantly monitors this log file for commands issued by other participants. These commands can then be decoded back into their original covert data via the decoding algorithm specified in Algorithm 4.8.

Algorithm 4.8 Algorithm for obtaining covert data from game commands

function decode(selected, (x, y), x_{max})
    size ← |selected|, z_1 = 0
    selected ← SORT-DESCENDING(selected)
    for i ∈ selected do
        z_1 ← z_1 + \left( \frac{i}{size} \right)
    end for
    z_2 ← (y × x_{max}) + x
    return (base2(z_1)||base2(z_2))
end function

This approach suffers from one minor drawback: replay logs for games from commercial studios are often stored in proprietary and undocumented formats that vary from game to game. However, reverse engineering the format of the replay logs is made significantly easier since Castle only issues MOVE or SET-RALLY-POINT commands. Therefore, we only need to understand how these commands are stored in replay logs. This can be done by simple techniques – e.g., sending a unit to the exact same location multiple times allows us to obtain the byte code used to signify the MOVE command, sending a unit to two locations in sequence (with each separated by a single pixel) allows us to obtain the bytes used to denote the (x, y) destination co-ordinates, etc. Further, for many popular real-time strategy games, these formats have already been reverse-engineered by the gaming/hacking community – e.g., 11 of the Top 20 real-time strategy games have decoded replay file formats.

Additionally, it is important to note that: (1) The overhead of decoding replay files is amortized over the entire set of users using that game as a cover, and (2) It is common for replay formats to be identical for real-time strategy games published by the same studio – e.g., most Microsoft real-time strategy games use the MGX replay format. Therefore, a working decoder for one game from one particular studio may work for all games from the same studio.
4.3.6 Bootstrapping Castle communication

In order to bootstrap covert communication, the following information needs to be shared between Castle users:

Castle user identity and configuration. For a covert channel to be established, a Castle user must first be able to find and join Castle game instances. Due to the absence of pre-established secrets, doing this in a secure way (i.e., a way that cannot also be used by the adversary censor to identify Castle game instances) is a currently an open research problem. We envision that current solutions such as BridgeDB [32] can be used for distributing game instance identities and configurations. In particular, BridgeDB may be used to distribute names and passwords of Castle game instances (in the case of games hosted on public servers) and IP address/port numbers of Castle games (in the case of peer-to-peer direct-connect game instances).

Castle map. In order to establish a covert communication channel, Castle users may also need to share a common Castle compatible map (that is used by the Castle game instance). While such maps might be quite large (in the order of a few MB), Castle provides a generic map generation script that is able to generate identical maps for all clients in the game with just a few bytes of configuration information.

Generally, to automate the process of Castle compatible map creation via a map editor, one needs a subroutine for creating buildings at specific locations on the map. Given this single subroutine, it is possible to automate the entire map generation process. In many map editors (e.g., map editors of 17 of the Top 20 best-selling real-time strategy games), we observe that such a subroutine only requires the automation of two clicks – one on a button to select the building type and one on the location at which the building is to be placed. For such editors we provide a generic Castle map creation automation script which only requires the following information for its building placement subroutine: the location of the button for the desired building type, the dimensions of the selected building type, and the available screen space. This information requires only a few bytes and allows users supplying the same parameters to generate identical maps. Additionally, it can be shared using the identity and configuration distribution mechanism.

4.3.7 Castle prototype implementation

We prototype on three games, each from a different publisher, to illustrate the extensibility of our approach.

- **0 A.D.:** An award-winning, free, open-source, and cross-platform real-time strategy game made available under the GPLv2+ license, by Wildfire Games.

- **Aeons:** A best-selling (in the top 2 grossing real-time strategy games of all-time), closed-source, Windows-based real-time strategy game from Studio X.

- **Conquerors:** A best-selling (in the top 5 grossing real-time strategy games of all-time) closed-source, Windows-based real-time strategy game from Studio Y.
Our prototype comprises of \( \sim 500 \) LOC and was coded in a combination of Python and AutoHotkey (desktop automation) \cite{125} scripts. It includes the following components:

**Custom map.** To test Castle, we created a custom game map for each of the three games. The map was comprised of \( n \) buildings packed as tightly as possible to facilitate our selection-based encoding. For 0 A.D., we created a map with \( n = 1600 \) buildings on a single game screen, while for Aeons and Conquerors, we were only able to have \( n = 435 \) and \( n = 416 \), respectively (owing to larger unit sizes). For all games, a region large enough to contain 16 bits of location data was left unoccupied. This is used to assign rally-point coordinates to the selected buildings.

Since 0 A.D. stores maps in a simple and readable XML format, the process of map creation was easily automated (via a Python script). This was not the case for Aeons and Conquerors which required manual generation of the map using the official GUI map editor. However, the current version of Castle comes with an easy to configure automation script to automatically generate Castle maps for many real-time strategy game requiring map generation via GUI editors (including Aeons and Conquerors).

**Data encoding and decoding.** Code for translating between covert data and in-game commands (and vice-versa) was written in under 200 lines of Python using the encoding and decoding described in Section 4.3.3. The output of the encoding code was a vector of buildings to be selected and a single \((x, y)\) coordinate.

**Desktop automation.** We used the open-source desktop automation tool, AutoHotkey, to execute the series of commands output by the encoding scheme. Since custom maps were used, the location of all buildings and units were known. As a result, selecting and commanding those indicated by our encoding program was straightforward.

**Reading recorded game data.** We implemented code that monitored the log file of commands issued (maintained by the game), for all games. For 0 A.D., this information was already made available in a simple to parse text file. In order to obtain this information for Aeons and Conquerors, the game replay file was parsed using replay-decoder tools and information made available by the gaming/hacking community. The file was then scanned to obtain each command as a vector of selected buildings and an \((x, y)\) coordinate. The commands were then decoded to retrieve the originally encoded covert data.

**Coordinate calibration.** The isometric perspective of the game screen posed a challenge during the decoding process. Specifically, the presence of a **viewing angle** meant that a sender may have intended to move a unit to the screen coordinate \((x_s, y_s)\), but the game actually logged the command as an order to move the unit to the game coordinate \((x_g, y_g)\), making this the command obtained by the receiver on decoding the move log. To avoid this, Castle goes through a one-time calibration process of mapping on-screen coordinates to coordinates as interpreted in the game. Note that the results of this calibration process can be shared across game clients that have the same resolution.
4.4 Evaluation

We evaluate Castle along three axes – security, extensibility, and performance. In Section 4.4.1 we consider security of the Castle by quantifying its resilience to the censor-adversary described in Section 4.2 and its ability to avoid the mismatches highlighted by Geddes et al. [111]. Next, in Section 4.4.2 we evaluate the extensibility of Castle – i.e., how easy is it to implement Castle over a closed-source game. Finally, in Section 4.4.3 we study throughput of Castle using the encoding scheme laid out in Section 4.3.

For the evaluation in Sections 4.4.1 and 4.4.2, we use our implementation of Castle with a building-based map, using SET-RALLY-POINT commands. The evaluation was performed on Windows 8.1 running AutoHotkey [125] for automation. The game was set up in direct connect mode – i.e., the two players were connected directly to each other via their IP address (rather than through the game lobby). Since both players were on the same (fast) university network, negligible effects of lag were experienced.

Castle was used to transfer a randomly generated (via /dev/urandom) 100KB binary file from one player to another. Network traffic generated by the game was captured using Rawcap (a command-line raw socket packet sniffer for Windows) with additional processing done using tcpdump.

We considered the impact of command rate (i.e., how long AutoHotkey waits between each issued command) and the impact of the maximum number of buildings selected \( k \) on the performance and security of Castle. For this we varied the command delays from 100 to 1000 ms/command and the number of selected buildings from 25 to 200.

In order to compare the traffic characteristics of Castle with characteristics of the standard game, we gathered network traces of regular 0 A.D. two-player games. These were also collected in a similar setting – i.e., with both players on the same university network and via direct connect. Ten traces were collected (one per game played). Each of the recorded games was between 20 and 60 minutes long.

In order to evaluate the extensibility of Castle, armed with a working implementation of Castle over 0-A.D., an undergraduate researcher was given the task of implementing Castle over the popular closed-source Aeons and Conquerors. Finally, to observe the impact of game-specific modifications, we evaluated the throughput of Castle over 0-A.D, Aeons, and Conquerors with and without any game-specific modifications, in the same settings described above.

4.4.1 Security Evaluation

We now perform an evaluation of Castle against the network adversary described in Section 4.2.

4.4.1.1 Resilience to network traffic attacks

Passive analysis. We first consider attackers with the ability to perform IP and port filtering, deep-packet inspection, and simple flow-level statistical analysis at line rate.
IP and port filtering: Since Castle actually uses an off-the-shelf implementation of the game application, the IP address and ports used by Castle are identical to that of the standard use of the game. This means that an adversary that triggers blocking based on the destination IP (e.g., the game server) or port number, will be forced to block all traffic to and from the game being used as the cover protocol.

In the event that the censor is willing to block all connections to dedicated game servers (often hosted by game publishers – e.g., Electronic Arts, Microsoft, Blizzard, etc.), clients may still utilize Castle in direct-connect (peer-to-peer) mode, forcing the censor into a game of whack-a-mole with Castle proxies hosted outside their jurisdiction. Further, users may also easily migrate Castle to another real-time strategy game whose game servers are unblocked.

It is also worth noting that blocking game flows is not without any costs to the censor, specifically with respect to political good will and PR internationally. For example, blocking all traffic for a given game, especially a popular title, may upset citizens and reflect poorly on Internet freedom within the censoring country [126, 127, 128].

Deep-packet inspection (DPI): When used with games that encrypt their communications, Castle is resistant to deep-packet inspection, since the censor cannot decrypt the stream of moves being made. However, since Castle works by issuing only generic commands (i.e., MOVE and SET-RALLY-POINT commands), it can easily be detected by DPI boxes if the game communicates commands in plain-text. Fortunately, it is generally recommended that real-time strategy games perform command channel encryption, making them resilient to DPI [113, 114, 115].

Flow-level statistical analysis: To quantify the resilience of Castle against flow-level attacks, several statistical tests and classifiers were employed. For each experiment, the Castle parameters that control the command rate and the maximum number of buildings selected were varied between 0 to 1000 ms and 25 to 200 buildings, respectively.

First, the Kolmogorov-Smirnov (KS) statistic was used to compare the similarity of human-game-generated traffic and Castle-generated traffic. Figures 4.3(a) and 4.3(b) reflects the KS similarity statistic on the packet size distributions of human- and Castle-generated games and Figures 4.3(c) and 4.3(d) does the same for inter-packet times. We make two observations from these plots: (1) There is a high variation in the flow-level features of legitimate (i.e., human-game-generated) traffic. We hypothesize that this is because the traffic generated by the real-time strategy game is strongly dependent on many parameters such as map and scenario type, strategies employed, and number of players. (2) Castle in many configurations, generates traffic that is well within this variance. We find that while restricting the maximum number of units per command to under 50 and the command rate to around 1 command/second, Castle generates traffic that is as similar to traffic generated by legitimate games.

Next, Castle was evaluated against several traffic fingerprinting classifiers. The goal was to evaluate the accuracy of classifiers, built for flow-level analysis, in distinguishing between Castle-generated and human-generated traffic.

First, each network capture was split into (20) one minute long chunks. For each experiment,
4.4. EVALUATION

Kolmogorov-Smirnov (KS) statistic on the distributions of packet sizes and times. The difference between Castle and the legitimate game flows is within the variance observed when comparing traffic between legitimate game flows.

classifiers were given 20 chunks of Castle-generated traffic at a specific configuration and 20 randomly selected human-game-generated chunks. Ten-fold cross validation was employed for splitting into training and testing sets.

Since, in our experiments, Castle was used for the purpose of file transfer, all traffic generated by it was in a single direction. This makes it trivially detectable by some fingerprinting classifiers which are heavily reliant on burst and direction features (e.g., k-NN[12], the Panchenko classifier [8] and OSAD[60]). We note that in a real deployment this directionality would not be an issue as there would be requests and responses from both sides.

Due to the directionality of traffic, traffic fingerprinting classifiers that ignored directional information were used. These included the Liberatore classifier [53], the Herrmann classifier [55], and an inter-packet timing classifier [116]. All classifier implementations were obtained from Wang’s open-source classifier archive [129]. The results of these experiments are illustrated in Figure 4.4. In general, the results indicate that Castle performs very well against packet size and timing classifiers,

Figure 4.3: Castle security evaluation: Impact of parameters on packet size and time distributions. Kolmogorov-Smirnov (KS) statistic on the distributions of packet sizes and times. The difference between Castle and the legitimate game flows is within the variance observed when comparing traffic between legitimate game flows.
with only the Herrmann classifier achieving an accuracy of over 60% against multiple configurations of Castle— i.e., only the Herrmann classifier achieved 10% higher accuracy than random guessing.

![Graphs showing classifier accuracy with varying parameters](image)

**Figure 4.4:** Castle security evaluation: Impact of parameters on classifier accuracy. The performance of Castle in various configurations against website fingerprinting classifiers.

**Active traffic manipulations.** In the face of active traffic manipulation attacks, such as probing, packet injection, and modification, Castle implemented over most commercial games faces little threat.

*Packet injection.* If Castle is implemented over a real-time strategy game with an encrypted and authenticated command channel, any packets injected by an unauthenticated source are dropped by the game-server. As a result, a probing adversary learns nothing about the Castle games running on the server.

*Packet modifications.* Most packet modification attacks are prevented by the presence of encrypted and authenticated in-game channels. Additionally, since Castle does not require any changes to the game or the hosting server, such attacks will always elicit the same response from both, legitimate game players and Castle users.

*Packet dropping and delaying.* Although most commercial real-time strategy games make use of UDP as a transport, the presence of reliability implemented in the application layer prevents any threats from adversaries that drop, or significantly delay packets in transmission. As a result, attacks (e.g., [111]) that result in denial-of-service for Castle users are not possible without also affecting legitimate game players.

### 4.4.1.2 Resilience to application layer attacks

Highly motivated censors may perform actions outside the realm of standard network traffic analysis and manipulation. We consider censors that may attempt to interact with the game server using custom game clients in order to reveal the identities of Castle users. Specifically, censors may
connect to game server lobbies to identify Castle games and try to join these games to learn the IPs of participating clients. For these cases, Castle provides several defenses based on features available in the game.

If the cover game supports the use of password-protected multi-player games, Castle proxies (i.e., hosts of Castle games) may configure Castle game instances to require users to authenticate using high-entropy passwords distributed using, for example, the BridgeDB mechanism [32]. Therefore censors without knowledge of the password are unable to join hosted games and learn the IP addresses of Castle users.

If the cover does not support the use of password-protected games, a Castle proxy may incorporate either (or, both) of the following defenses against these adversaries: (1) The proxy may use standard game maps rather than custom-made Castle game maps. This allows Castle instances to blend in with legitimate game instances, making it harder for the censor to identify which games to join. However, this comes at the cost of lower throughput since there are typically fewer units in standard game maps. (2) The proxy may still use a BridgeDB-like mechanism for password distribution and require that any Castle client makes the moves corresponding to the supplied password in order to receive proxying services. In the event that a client does not supply this password within some period of time, the Castle proxy may continue playing the game using a standard AI. Therefore, even a censor that may enter games is unable to distinguish between Castle games and legitimate games.

**Deniability and ease of distribution.** In addition to being resilient to computational attacks, Castle also has the advantage of being a covert channel that is largely implemented with off-the-shelf software components with only a few hundred lines of code dedicated to encoding and desktop automation scripting. Desktop automation tools are already commonly used by gamers; the game and game-specific mods (e.g., replay decoder and map editor) are widespread enough to warrant little suspicion from censors since (e.g., Aeons is installed by millions of users worldwide). Castle’s small code base also makes it easy to distribute via hard to block asynchronous methods – e.g., through collage [130], email, instant messaging, etc.

### 4.4.1.3 Avoiding covert channel pitfalls

Geddes *et al.* highlight three key mismatches between covert channels and cover traffic which make these look-like-something circumvention tools detectable to external observers [111]. Here we discuss how Castle avoids each of these three mismatches.

**The architecture mismatch.** Games provide agility in terms of architecture that few other channels provide. They often operate in client-server mode on publisher-hosted game servers and in peer-to-peer mode in direct-connect multi-player games. Our proxying approach can operate in whichever mode is the dominant, and in the presence of blocking can even shift (e.g., from client-server mode to peer-to-peer mode).

**The channel mismatch.** While game data is typically communicated over a UDP channel, it is unresilient to packet loss unlike other UDP-based channels (e.g., VoIP), thus clients come with the
ability to handle packet losses and retransmissions. Further, they also guarantee in-order delivery and processing of sent data. This makes it useful as a covert channel for proxied TCP connections which require reliable transmission. Therefore, attacks that allow the censor to drop traffic to levels which are tolerable to legitimate players (but intolerable to Castle users) are not possible.

The content mismatch. Content mismatches arise when the content being embedded in the covert channel changes the flow-level features of the channel. Since the flow-level features of real-time strategy games are strongly dependent on many parameters (identified above), they are highly variable. We have shown that Castle, under every configuration, generates traffic that is well within this variance.

### 4.4.2 Performance Evaluation

Without any game-specific modifications, Castle offers performance amenable to transfer of textual data (e.g., tweets, e-mail, news articles) and even bootstrap higher bandwidth secure communication channels (e.g., for distribution of Tor Bridge IP addresses).

#### 4.4.2.1 Castle throughput

The throughput achieved by Castle is dependent on two certain game characteristics – maximum number of units on a game screen and maximum number of selectable units in a single command.

- **Maximum units per game screen**: Depending on the size of the units used within the game and the layout of the game screen, the number of units that may be placed within a Castle map for the game varies. For example, as illustrated in Table 4.2, 0-A.D. is able to fit up to 1600 units on a Castle map, while Aeons and Conquerors allow only up to 435 and 416 units, respectively.

- **Maximum selectable units per command**: Some games impose limitations on the number of units that may be commanded at once. For example, 0-A.D. allows only up to 200 units/command and Conquerors allows only up to 40 units/command.

---

2 The success of the voices feeds during the Arab Spring shows that in some situations textual data is enough to get information out.
4.4. EVALUATION

Since Castle is able to send up to \( \sum_{k=1}^{k=n} \log_2 \left( \frac{n}{k} \right) \) bytes per command on average, these parameters directly affect its throughput. Given the game specific parameters, we are able to achieve averages of approximately 65 bytes/command for 0-A.D., 39 bytes/command for Aeons, and 14 bytes/command for Conquerors.

In addition, throughput is also dependent on the time required by the desktop automation tool to perform the actions required to issue a command (i.e., click each unit to be selected and click the target coordinate) and the time delay issued between successive commands.

To illustrate the effects of these parameters within a particular game, in Figure 4.5 we see their effect on Castle transfer rates when implemented over 0 A.D. Specifically, Figure 4.5(a) shows the effect of increasing the maximum number of buildings selected in a single command and Figure 4.5(b) demonstrates the effect of increasing the delay between commands.

At the average performance configurations for 0-A.D., Aeons, and Conquerors, vanilla Castle requires 52, 77, and 238 seconds for transferring a short 10KB file, respectively. This is suitable for asynchronous communication and bootstrapping higher bandwidth communication channels.

![Figure 4.5: Castle performance evaluation: Impact of parameters on throughput.](image)

(a) Effect of maximum number of buildings selected per command (at \( \approx 100 \) ms delays/command)

(b) Effect of time delays per command (at \( \leq 200 \) buildings selected/command)

4.4.2.2 Improving Castle throughput

There are several approaches to improve the throughput of castle.

**Parallel requests:** Since modern real-time strategy games allow eight or more players to participate in a single multi-player game, it is possible for one censored user to encode content requests to as many as seven (or more) proxies in parallel – achieving at least a seven fold increase in throughput. This is particularly useful in the context of web data, where requests are easy to parallelize.

**Game specific enhancements:** Many real-time strategy games offer features that are not universal. For instance, many games provide trigger controls to map designers – i.e., a feature that allows map designers to specify responses to player actions (if a player performs action \( x \), action \( y \) happens to unit \( z \)). Such features allow Castle to encode significantly more data than currently possible.

\(^3\)We refer to the original Castle design as described in Section 4.3 (without any additional enhancements) as vanilla Castle.
– e.g., Castle could use a hierarchical encoding structure if camera motion actions are permitted in trigger systems. Other games provide significantly more comprehensive replay information and include preserving the order of clicks performed by opponents. This allows castle to achieve significantly more Bytes per command \(O(\log_2 P(n, k))\) than it currently does \(O(\log_2 C(n, k))\). An example of such a non-universal feature used to improve Castle’s throughput is demonstrated for Aeons in Section 4.4.2.3.

**Content compression:** Castle proxies may improve performance by compressing requested content before encoding. In the context of web data, the proxies may also pre-render and compress content before sending to the Castle receiver (e.g., as was done by the Opera mobile browser [132]).

### 4.4.2.3 Game-specific enhancements for Castle

In this section, we show that the performance of Castle can be improved significantly through simple game-specific tweaks. To be able to observe the impact of these game-specific modifications, Aeons was used as the channel for vanilla Castle and Castle with Aeons-specific modifications. The game-specific modifications were introduced and implemented for Castle in just under three hours by an undergraduate researcher.

The low throughput of Castle over Aeons was because Aeons had larger units than 0 A.D., thereby allowing players to place only 435 units within a single screen (as opposed to 1,600 for 0 A.D.). As a result, the throughput of vanilla Castle was only \(\approx 38\) bytes/command (i.e., \(\approx 130\) bytes/second) – i.e., with the maximum command rate of AutoHotkey and selection of up to 435 units/command.

A quick investigation into the Aeons replay mode and save-game files revealed that even the selection of a single unit was communicated over the network and logged by other players. We exploit this fact by creating a set of \(2^m\) units (256 in our case) and mapping each unit to an \(m\)–bit value (i.e., a byte). We then sequentially transfer the data byte-by-byte via selecting the unit corresponding to the byte to be encoded.

This encoding allowed AutoHotkey to issue commands at a significantly faster rate than before (a command was now just a single mouse click, as opposed to up to 435 key presses and clicks). At AutoHotkey’s fastest mouse click rate and \(m = 8\), this encoding achieves a throughput of up to 3 KByte/second. However, in order to more closely mimic the command rate and traffic generated by a skilled human player, a delay of 3 ms per command is added.

In Figure [4.6](#), we show the effect of this game-specific modification on the throughput of Castle. From the same figure, we can also observe the effect of varying the total number of units with vanilla Castle and the Aeons-specific version of Castle. We see that increasing \(n\) results in a linearly increasing throughput for vanilla Castle, and a logarithmically increasing throughput for Aeons-specific Castle. However, because the cross-over point of these functions is higher than the game allows, Aeons-specific Castle always achieves better throughput for Aeons.
4.5. Extensibility of Castle

In order to evaluate the extensibility of Castle to new real-time strategy titles, we considered the time required and the development procedure used by an under-graduate researcher to complete a basic port of Castle over two extremely popular (over 8.5 million copies sold) closed-source real-time strategy games from two different development studios – Aeons and Conquerors.

Castle attempts to be easily adapted to many real-time strategy games by only utilizing the common command structure for encoding and replay files for decoding. As a consequence, it was possible to port Castle to Aeons and Conquerors in under 6 hours per game. The three main phases for porting Castle to a new real-time strategy game are map creation, configuring the automation toolkit, and decoding replay files.

Map creation. In some real-time strategy games where game maps are stored in easy to read formats (e.g., 0-A.D.), maps for use with Castle can be generated via simple scripts. In others which use proprietary map storage formats (e.g., Aeons and Conquerors), the developer is required
to manually place units at specific (known) locations on the game map. In such cases, to reduce
the effort required for this time consuming process, Castle currently provides an easy to configure
AutoHotKey script to automatically generate maps via desktop automation and the GUI map
editor of any real-time strategy game.

**Desktop automation.** To allow Castle to execute commands within the game, desktop automa-
tion tools have to be integrated with the real-time strategy game. During this process the developer
is required to supply configuration parameters including maximum number of selectable units, click
sending mode (AutoHotKey provides four modes. Selection of a mode is dependent on the type of
application and DirectX version), window title, and a suitable inter-click speed (*e.g.*, Conquerors
blocks clicks from Castle in its fastest configuration).

**Decoding replay files.** Finally, in order to decode data sent by a Castle client, Castle needs to
be able to retrieve data stored in the form of in-game commands in game replay files. Fortunately,
replay decoders are already available in the hacking/gaming community for many popular games
(*e.g.*, Aeons). For other games without an available decoder (*e.g.*, Conquerors), gaming and hacking
forums such as [133, 134, 135] provide techniques and support for building a decoder. Additionally,
the process is made simpler since it is sufficient for the developer to be able to extract `MOVE` or
`SET-RALLY-POINT` commands (rather than needing the ability to decode any command stored in a
replay file). Finally, since most real-time strategy games from a studio use the same replay format,
the overhead of decoding replay files is amortized over the entire set of users using real-time strategy
games from the same studio as a cover.

### 4.6 Castle and Other Look-like-something Circumvention Tools

In this section we compare the design methodology, extensibility, and performance of Castle with
the state-of-the-art in *look-like-something* circumvention systems: Rook [120], FreeWave [35], and
Skypemorph [34].

**Design methodology and adversary model.** Like Castle, both FreeWave and Rook actually
use their cover protocols (VoIP and video games, respectively) – *i.e.*, they insert covert data via
the application layer, rather than directly at the transport layer (as is done by Skypemorph). As a
result, standard mimicry detection attacks such as IP/Port filtering and active probing are unable
to distinguish the use of the covert channel from the cover channel (while they succeed against
Skypemorph [111, 110]).

While both FreeWave and Castle rely on the fact that their main communication channels are
encrypted, Rook does not. Rather, Rook focuses on achieving steganographic security (resulting in a
much stronger adversary); even if the adversary is able to observe the unencrypted communications
between a Rook server and client, it is still unable to distinguish the usage of Rook from the cover
video game. This is not the case with Castle or FreeWave – *i.e.*, an adversary that is able to
observe unencrypted communication between the proxy and the client is easily able to distinguish
the covert channel from the cover channel.
Although Castle, Rook, and FreeWave all use UDP communication channels, FreeWave is unable to avoid detection by active attacks which perturb the network traffic by delaying or dropping packets. This is a result in a mismatch of reliability requirements between the cover protocol (multimedia VoIP) and the covert channel data which demands higher reliability. In contrast, Castle and Rook leverage video games as a cover protocol where reliability is built into the application layer (by default in Castle and by the covert protocol in Rook).

**Extensibility.** Since Castle, Rook, and FreeWave insert covert data via the application layer, they are extensible to varying degrees. Skypemorph on the other hand is built to mimic the Skype protocol, and is therefore not extensible. As demonstrated in Section 4.5, Castle is easily extensible to any real-time strategy game which has (1) a map editor, (2) a MOVE or SET-RALLY-POINT command, and (3) a replay file decoder. In contrast, extensibility of Rook and Freewave is hindered by: requiring a deep understanding of the internal networking protocol used by the cover channel (Rook), or the absence of a large number of cover applications to extend to (FreeWave).

**Performance.** In terms of throughput, both VoIP based covert channels – FreeWave and Skypemorph perform better than Castle and Rook. Skypemorph is able to achieve a covert data transmission rate of 34 KBps, while FreeWave achieves 2.4 KBps. The large difference between the two systems should be attributed to the significantly stronger adversary model used by FreeWave (FreeWave is built to be secure against active probing). The throughput achieved by Castle is dependent on the characteristics of the game being used as a cover channel. In our experiments considering three different cover video games, we observed covert data transfer rates in the range of 42 - 320 Bps without any modifications to Castle. With game specific modifications, we were able to achieve up to 435 Bps. Since Rook focuses on steganographic security (a much more powerful adversary than Castle), it also suffers from lower throughput – *i.e.*, between 3 and 5 Bps.

### 4.7 Conclusions

In this chapter, we presented Castle, a general approach for creating covert channels using real-time strategy games as a cover for covert communications. We demonstrate our approach by prototyping on three different games with minimal additional development overhead and show its resilience to a network adversary.

We argue that the popularity, availability, and generic functionalities of modern games make them an effective circumvention tool in the arms-race against censors. Specifically, our results show that Castle is:

- **Portable and Extensible:** Incorporating new closed-source games as covert channels for Castle requires only a few hours of developer time – including the addition of title-specific enhancements for increased throughput.

- **Secure:** Castle is resistant to attacks such as IP/port filtering and deep-packet inspection since it actually executes the game application. More complicated and expensive attacks such
as traffic analysis attacks are avoided due to the high variability of standard game flows. In addition, Castle is also resilient against active and application-layer attacks.

- **Usable:** Even without any game-specific modifications, Castle is able to provide throughput sufficient for transfer of textual data and bootstrapping higher-bandwidth channels.

The results presented in this work motivates two independent future research directions. First, Castle demonstrates that portability is possible in circumvention tools. Therefore, extending our work to different classes of applications which may enable higher throughput rates may yield a more powerful defense against censorship. Second, integrating the Castle approach into platforms to make it usable to users e.g., via a Web browser plug-in or integration with the suite of Tor Pluggable Transports [7].
Chapter 5

Measuring the Impact of Users Abusing Anonymity

In this chapter, we focus on measuring server-side discrimination faced by legitimate Tor users as a consequence of abusive behavior from malicious users of the network. The results presented in this chapter have previously appeared as part of the following articles: [29].
5.1 Introduction

Anonymity systems like Tor provide a useful service to users who wish to browse the web without revealing their intended destination to any local monitoring, or their network-layer identity to the final destination. However, as Tor has increased in scale and usage, tensions have emerged between Tor users and online service providers. Specifically, service providers claim that the anonymity provided by Tor is often used maliciously for spamming, vulnerability scanning, scraping, and other undesired behavior (e.g., [37]). As a result, Tor users now face differential treatment (e.g., needing to solve CAPTCHAs before receiving services) and even outright blocking [28].

At its core, the problem is that in return for anonymity, each Tor user shares their reputation with other users. As a result, the malicious actions of a single Tor user can lead IP abuse blacklists to include IP addresses used by Tor exit relays. Consequently, websites and content providers treat even benign Tor users as malicious. In our work, we characterize aspects of the conflict between users desiring anonymity and websites aiming to protect themselves against malicious Tor traffic. We investigate the nature of traffic that exits the Tor network and is undesired by online service providers. We also actively measure various forms of discrimination performed against Tor users.

Challenges. We grapple with two key challenges: First, measuring Tor traffic is antithetical to the goals of the anonymity system and presents ethical challenges. Second, defining and identifying undesired or abusive network traffic is challenging as opinions vary and encryption can render inspection of traffic infeasible. We address both challenges by focusing on the receivers’ reactions to Tor traffic rather the traffic itself. Specifically, we take measurements of server responses to Tor traffic, both synthetic (§5.4) and user-driven (§5.5). These datasets not only allow us to observe the effects of undesired traffic without directly observing it, but also provide an operational definition of undesired traffic: the traffic that leads to rejecting of Tor users. This operationalization allows us to sidestep debates over what constitutes abuse and to focus on the subset of undesired Tor traffic that has impacted operators and users.

Measuring the prevalence of discrimination faced by users requires exercising multiple aspects of websites and inspecting them for subtle forms of discrimination (e.g., CAPTCHAs and interaction based discrimination) in addition to outright blocking. To address this issue and accurately measure discrimination against users, we go beyond the prior work of Khattak et al. and develop a crawler capable of exercising the search and login features of websites. Taking measurements of real Tor traffic required the creation and deployment of a privacy-sensitive logging approach for our own Tor exit relays. We also consider aspects of Tor exit relays that make them more susceptible to blocking. This is done by deploying several Tor exits with varied configurations and monitoring the reactions they produce.
5.2 Background and Related Work

5.2.1 Tensions between Tor and online services.

Tor is a low-latency onion routing network with over 2M daily users and over 7K supporting servers [5]. While the anonymity provided by Tor is lauded by proponents of Internet freedom, it can also provide a cloak for malicious network activities. Indeed, CloudFlare reported that 94% of the requests from the Tor network are “malicious”, consisting of comment spam, scanning, and content scraping [37]. According to a report published by Distill networks, 48% of Tor requests are malicious, higher than non-Tor attacks (38%) [38]. A study of the Sqreen application protection service found that connections through Tor are responsible for ≈30% of all attacks on their customers, including password brute force attacks, account enumerations, and fraudsters [136]. As per Akamai’s State of the Internet report, an HTTP request from a Tor IP address is 30 times more likely to be a malicious attack than one from a non-Tor IP address [137]. Imperva-Incapsula found that in a period of 2.5 weeks, 48.53% of the attack requests came from Tor [138]. However, the majority of these attack sessions were originated from the well-known DDoS bots and bad clients, which can be identified using approaches other than IP reputation. Removing the attacks from the known attackers, the attack sessions originating from Tor went down to 6.78%, which is comparable the attacks coming from Ireland (5.45%).

Different services have reported similar types of attacks from Tor. The three most common attacks from Tor to Akamai’s services were automated scanning (path scanning and vulnerability scanning), SQL injection and cross-site scripting attacks [137]. IBM reports that SQL injection, automated scanning and DDoS are the most common attacks from Tor [139]. Sqreen found authentication attacks (brute force attack on a specific user account, or accounts enumeration), path scanning and SQL/NoSQL injections are likely to originate from Tor [136]. Despite reports claiming a higher likelihood of malicious traffic from Tor, there have been debates about the correctness of their inference methods. For instance, Perry, writing for the Tor Project’s blog, questions whether CloudFlare’s methods considered as malicious all traffic from an exit relay that ever sent any malicious traffic [140]. While websites might be tempted to blacklist all Tor IPs in a proactive attempt at security, doing so could cause a loss in revenue. Akamai’s report highlights that Tor users are just as likely to make purchases from revenue generating websites, as non-Tor users [137].

5.2.2 Blocking and Filtering of Tor.

Many government censors around the world block access to Tor [24], the subject of numerous measurement studies [141, 142, 143, 144, 145]. However, such government censorship blocks access to the Tor entry nodes, which is different from server-side Tor blocking, which blocks access from the Tor exit nodes.

Khattak et al. is the only systematic measurement study of server-side Tor blocking [28]. They show that at least 1.3 million IP addresses blocked Tor at the TCP/IP layer and 3.6% of the Alexa
Top 1,000 websites blocked Tor at the HTTP layer in 2015. At the TCP/IP layer, the hosting services GoDaddy and Dreamhost are among the top five Tor blockers. CloudFlare blocks access at the HTTP layer. Our work extends the work of Khattak et al. by additionally measuring the blocking of login and search functionality. We find a higher rate of blocking (20.03%) than Khattak et al. (3.6%). We demonstrate that Khattak et al.’s headless crawler underestimates the blocking rate (Figure 5.10).

However, such technical measurements fail to estimate the actual impact of blocking on the Tor users. Tor users have taken steps to characterize the impact of blocking by crowdsourcing several lists of websites that block Tor [146, 147]. To understand the impact of blocking on Tor users, we measure the number of failed requests to Alexa top 1M web pages at the exit level using privacy sensitive logging on our exits.

5.3 Our Deployed Exits

To aid our study of user discrimination due to abuse, we deploy and use data from ten of our own exits in addition to current records about pre-existing Tor exits.

<table>
<thead>
<tr>
<th>Relay Pseudonym</th>
<th>Max. BW</th>
<th>Exit Policy</th>
<th>Num.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large-Default</td>
<td>61 MBps*</td>
<td>Default</td>
<td>2</td>
</tr>
<tr>
<td>Medium-Default</td>
<td>10 MBps</td>
<td>Default</td>
<td>2</td>
</tr>
<tr>
<td>Medium-RR</td>
<td>10 MBps</td>
<td>RR</td>
<td>2</td>
</tr>
<tr>
<td>Small-Default</td>
<td>2 MBps</td>
<td>Default</td>
<td>2</td>
</tr>
<tr>
<td>Small-RR</td>
<td>2 MBps</td>
<td>RR</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.1: Configurations of our deployed exit relays.
*The large exits’ policy allows for unlimited bandwidth usage. We provide the maximum bandwidth achieved during the study period.

We vary the bandwidth allocated to and exit policy used by our exits in order to understand the impact of relay characteristics on email complaints, blacklisting, and discrimination. Bandwidth allocated to relays were 2 MBps (small exits), 10 MBps (medium exits), and unlimited (huge exits). In total, our deployed relays handled over 3% of all Tor exit traffic during their deployment. The exit policies were varied to be either the Tor default policy or the “Reduced-Reduced” policy. The default policy [148] allows all ports except those associated with spam (25, 119), attacks (135–139, 445, 563), or peer-to-peer file sharing (1214, 4661–4666, 6346–6348, 6699, 6881–6999) including some adjacent ports (6349–6429). The Reduced-Reduced (RR) exit policy, designed to avoid blacklisting, only allows commonly used ports [149]. Our relay configurations are summarized in Table 5.1.

Breaking down of the most used ports on our exit relays based on their exit-policy, we see that web-traffic accounts for 98.88% of all traffic passing through our RR policy exits. In contrast, our default policy exits have much more application/port diversity: only 31.36% of observed traffic is
5.4 SYNTHETIC DISCRIMINATION MEASUREMENTS

HTTP(S). We measure this using our privacy-sensitive logging described in Section 5.5.

5.4 Synthetic Discrimination Measurements

To quantify the number of websites practicing discrimination against Tor, we perform crawls looking both at front-page loads, as in prior work [28], and at search and login functionality. We crawl the Alexa Top 500 web-pages from a control server and a subset of Tor exit relays. These crawls identify two types of discrimination against Tor users: (1) the Tor user is blocked from accessing content or a service accessible to non-Tor users, and (2) the Tor user can access the content or service but only after additional actions not required of non-Tor users – e.g., solving a CAPTCHA or performing two-factor authentication.

This experimentally derived dataset allows us to measure the number of popular websites practicing discrimination in various forms: front-page loads, searches, and logins.

5.4.1 Crawler Design

We developed and used a Selenium-based interactive crawler to test the functionality of websites. Three types of crawls were performed: (1) Front-page crawls attempt to load the front page of each website. The crawl was repeated four times over the course of six weeks. (2) Search functionality crawls perform front-page loads and then use several heuristics to scan for the presence of a “search” feature. If the feature is found, it submits a pre-configured search query. Our crawler found and tested the search functionality of 243 websites from the Alexa Top 500. The search functionality crawl was performed once. (3) Login functionality crawls loads front-pages and then scans them for the presence of a “login” feature. If the feature is found and credentials for the webpage are available, the crawler authenticates itself to the site (using Facebook/Google OAuth when site-specific credentials were unavailable). We created accounts on OAuth compatible websites prior to the crawl. Since the created accounts had no prior history associated with them, they were unlikely to be blocked as a result of unusual behavior. Our crawler found and tested the login functionality of 62 websites from the Alexa Top 500. The login functionality crawl was performed once.

The crawler records screenshots, HTML sources, and HARs (HTTP ARchives) after each interaction. Our interactive crawler improves upon previous work in several ways. First, since it is webdriver driven, it uses the Firefox browser, and incorporates several bot-detection avoidance strategies (e.g., rate limited clicking), it avoids the bot-based blocking that make performing page-loads via utilities such as curl or non-webdriver libraries such as urllib unsuitable. Second, its ability to interact with websites and exercise their functionality allows us to identify cases where discrimination only occurs when users interact with the website beyond viewing the front page – e.g., www.tumblr.com serves Tor users CAPTCHAs only after they submit a search query, and www.imdb.com blocks Tor users when they attempt to log in.
5.4.2 Relay selection

We randomly selected 100 exit relays from the set of all exit relays that were able to support HTTP(S) connections (i.e., accept connections to ports 80 and 443). Figure 5.1 shows the distributions of characteristics (bandwidth, uptime, and exit-policies) of our sampled relays compared to the set of all suitable exit relays. In addition to these randomly sampled relays, we also conducted crawls through our own relays described in Table 5.1 and a university hosted control server.

Since our crawls were performed over a six week period, several of the selected exit-relays intermittently went offline. 0, 12, 19, and 28 were offline during crawls 1, 2, 3, and 4, respectively. We account for the resulting page-load failures by excluding the failures from our analysis. Figure 5.2 shows the detailed availability during our crawls.

5.4.3 Identifying discrimination

In each of our experiments, we simultaneously performed crawls exiting through all online sampled exits and a university hosted control server. To identify discrimination of a selected exit relay, we first rule out cases of client and network errors through HAR file analysis. We use the HAR files
5.4. SYNTHETIC DISCRIMINATION MEASUREMENTS

Figure 5.2: Synthetic measurements. Availability of selected relays during the period of our crawls. Blue blocks indicate that no descriptor was published by the relay during the corresponding 24 hour period.

to verify, for each page load, that (1) the requests generated by our browser/client were sent to the destination server (to eliminate cases of client error) and (2) our client received at least one response from the corresponding webpage (to eliminate cases of network errors). If, for a given site, either the control server or the selected exit relay did not satisfy both these conditions, we did not report discrimination due to the possibility of a client or network error.

Next, we compare the crawler recorded screenshots of the control-server and each selected exit relay using perceptual hashing (pHash)\(^{[150]}\), a technique that allows us to identify the (dis)similarity of a pair of images. We report images with high similarity scores (pHash distance < .40) as cases where no discrimination occurred and images with high dissimilarity (pHash distance > .75) as cases of discrimination, while flagging others for further inspection. The thresholds were set so that only pages with extreme differences in content and structure would be automatically flagged as cases of discrimination, while similar pages were automatically flagged as cases of non-discrimination. In general, minor changes in ads/content (e.g., due to geo-location changes) do not result in flagging. The thresholds were set using data obtained from a pilot study (Figure 5.3).

Then, cases where HTTP error codes were issued to exit relays for requests that were successfully
Figure 5.3: Synthetic measurements: Results of pilot study to identify pHash distance thresholds for automatically identifying cases of (non) discrimination. 500 randomly chosen samples (i.e., pairs of control and exit relay screenshots of the same website) were manually tagged and the pHash distances were computed. Based on the above distribution of pHash distances, we classified instances with pHash distances < .40 as “non-discrimination” and instances with pHash distances > .75 as “discrimination”. Instances having pHash distances in the .40 to .75 range were manually inspected and tagged.

loaded with a 200 status by our control server are classified as discrimination. Finally, we manually tag the screenshots of remaining cases to identify more subtle discrimination – e.g., a block-page served with a 200 status.

5.4.4 Results

Table 5.2 summarizes the main results of our three types of crawls over compatible websites in the Alexa Top 500. Here, we show the fraction of interactions on which discrimination was detected. We find that 20.03% of all Alexa Top-500 (A-500) website front-page loads showed evidence of discrimination against Tor users, compared to 17.44% of the search compatible (S-243) and 17.08% of the login compatible (L-62) website front-page loads. Additionally, when considering blocking of the search and login functionality, we see a 3.89% and 7.48% increase in discrimination, respectively.

<table>
<thead>
<tr>
<th>Websites</th>
<th>Interaction</th>
<th>Discrimination observed</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-500</td>
<td>Front page</td>
<td>20.03%</td>
</tr>
<tr>
<td></td>
<td>Front page + Search</td>
<td>21.33% (+3.89%)</td>
</tr>
<tr>
<td>S-243</td>
<td>Front page</td>
<td>17.44%</td>
</tr>
<tr>
<td></td>
<td>Front page + Search</td>
<td>21.33% (+3.89%)</td>
</tr>
<tr>
<td>L-62</td>
<td>Front page</td>
<td>17.08%</td>
</tr>
<tr>
<td></td>
<td>Front page + Login</td>
<td>24.56% (+7.48%)</td>
</tr>
</tbody>
</table>

Table 5.2: Fraction of interactions blocked from 110 exits. A-500 denotes the Alexa Top 500 websites, S-243 denotes the 243 search compatible websites, and L-62 denotes the 62 login compatible websites.

Figure 5.4(a) shows the distribution of discrimination (for any interaction) faced by relays from
websites in the Alexa Top 500. We find that no relay is discriminated against by more than 32.6% of the 500 websites, but 50% of the exit relays are discriminated against by more than 27.4% of the 500 websites. Figure 5.4(b) shows the distribution of discrimination performed by websites against Tor exit relays. Here, we see that 51% of the websites perform discrimination against less than 5% of our studied exits, while 11% of websites perform discrimination against over 70% of our studied exits.

Figure 5.12 shows the sites performing the most discrimination for each interaction. We observe a few websites (e.g., www.meetup.com) perform blocking of Tor users from even loading the website’s front-page. Other sites, such as www.imdb.com and www.tumblr.com, let Tor users access the front-page, but prevent them from performing searches and logging in.

![CDF of sites](image1)

(a) Distribution of discrimination faced by relays

![CDF of relays](image2)

(b) Distribution of discrimination performed by websites

Figure 5.4: Synthetic measurements: Distribution of discrimination by Alexa Top 500 websites against 110 exit relays.

We now examine various factors associated with Tor discrimination. Since we did not (and any many cases cannot) randomly assign these factors to websites or relays, these associations may not be causal.

**Hosting Provider.** Figure 5.5 shows the fraction of relays discriminated against by websites hosted on four of the six most used hosting platforms. We find that Amazon hosted websites (Figure 5.5(a)) show the most diversity in discrimination policy, which we take as indicative of websites deploying their own individual policies and blacklists. In contrast, CloudFlare (Figure 5.5(b)) and Akamai (Figure 5.5(c)) have several clusters of websites, each employing a similar blacklisting policy. This pattern is consistent with CloudFlare’s move to allow individual website administrators to choose from one of several blocking policies for Tor exit relays [37]. Finally, we see 80% of China169 hosted websites (Figure 5.5(d)) perform discrimination against at least 60% of our studied relays. Interestingly, previous work has shown that the Great Firewall performs active probing of Tor bridges from within the China169 network for the purpose of blocking connections to and from the Tor network [142].

**Relay Characteristics.** Our analysis of the association between exit relay characteristics and the discrimination faced by them found no significant correlations when accounting for relay-openness (fraction of ports that the exit relay is able to service requests for) or for the age of the relay on the discrimination faced by it. However, there was a small positive correlation (Pearson correlation...
coefficient: .147) between the relay bandwidth and discrimination faced, but the result was not statistically significant (p-value: 0.152). These results are graphically presented in Figure 5.6. We further analyze the impact of relay characteristics on discrimination performed by websites using popular hosting providers (Figure 5.7). We find that only Amazon has a statistically significant positive correlation between discrimination observed and relay bandwidth (Pearson correlation coefficient: .247, p-value: .015).

Service Category. We now analyze how aggressively four different categories of sites—search engines, shopping, news, and social networking—discriminate against Tor exit relays. We categorize sites using the McAfee URL categorization service [151]. We find that search engines are the least likely to discriminate against exit relays with 83% of all search engines discriminating against less than 20% of our studied exit relays, compared to 32% of shopping sites, 53% of news sites, and 30% of social networking sites. Social networking sites are found to be the most aggressive—with 50% of them blocking over 60% of the chosen relays. The results are illustrated in Figure 5.8.

The Evolution of Tor Discrimination. We now focus on discrimination changes over time.
5.4. SYNTHETIC DISCRIMINATION MEASUREMENTS

![Figure 5.7: Synthetic measurements: Impact of relay characteristics on discrimination performed by websites hosted by popular hosting providers](image1)

Figure 5.7: Synthetic measurements: Impact of relay characteristics on discrimination performed by websites hosted by popular hosting providers

![Figure 5.8: Synthetic measurements: Distribution of discrimination performed by websites in four website categories.](image2)

Figure 5.8: Synthetic measurements: Distribution of discrimination performed by websites in four website categories.
For this experiment, we conducted four crawls via our own ten exit relays to the Alexa Top 500 websites. We conducted crawls on day -1 (the day before the relay was assigned its exit-flag), day 0 (as soon as the relay was assigned its exit flag), and once a week thereafter. Table 5.9 shows the fraction of websites found to be discriminating against each exit set during each crawl. Increases in discrimination are observed when the exit flag is assigned. Some of this can be attributed to our improved crawling methodology deployed on Day 0 (the Day -1 crawl utilized the crawler from Khattak et al.), although it must also be noted that the IP addresses used by our exit relays were never used by other Tor exit relays in the past and were not present on any of our studied commercial blacklists before Day 0, while they were present immediately after the exit-flags were assigned.

![Figure 5.9: Synthetic measurements: Percentage of discriminating page loads observed by each of our deployed relays over time.](image)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Day -1</th>
<th>Day 0</th>
<th>Wk. 2</th>
<th>Wk. 3</th>
<th>Wk. 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large-Default</td>
<td>NA</td>
<td>17.0</td>
<td>19.0</td>
<td>21.1</td>
<td>25.4</td>
</tr>
<tr>
<td>Medium-Default</td>
<td>9.4</td>
<td>20.5</td>
<td>24.4</td>
<td>25.6</td>
<td>24.8</td>
</tr>
<tr>
<td>Medium-RR</td>
<td>9.9</td>
<td>18.3</td>
<td>24.1</td>
<td>22.7</td>
<td>24.7</td>
</tr>
<tr>
<td>Small-Default</td>
<td>9.3</td>
<td>20.3</td>
<td>20.9</td>
<td>23.9</td>
<td>23.6</td>
</tr>
<tr>
<td>Small-RR</td>
<td>9.4</td>
<td>20.5</td>
<td>20.7</td>
<td>25.7</td>
<td>25.3</td>
</tr>
</tbody>
</table>

Table 5.3: Percentage of discriminating page loads for each set of deployed relays. The results are presented in graphical format in Figure 5.9.

The high amount of discrimination observed on our Day-0 crawl for all exit relays is indicative of proactive discrimination against Tor exit relays. Our results do not indicate differences in amount of discrimination based on the relay category.

**Measurement Methodology.** We now measure the impact of changes in our discrimination identification methodology compared to previous work by Khattak et al. [28]. The key differences between the methodologies are: (1) The measurements conducted by Khattak et al. are limited to identifying front-page discrimination. Our crawler also tests search and login interactions. The impact of this feature is presented in Table 5.2. (2) Khattak et al. identify discrimination using
the difference in HTTP status codes returned by the control and test nodes. This method is prone
to underestimating discrimination due to the inability to detect block pages that return a HTTP
200 OK status code. Our method relies on screenshot differences and HTTP status codes as signal
for discrimination. As a result, we are able to detect discrimination performed by sites such as
livejournal.com, hdfc.com, and glassdoor.com. (3) Khattak et al. rely on sending HTTP
requests for front-pages of websites using the python urllib2 library. Although they modify the
user agent of their crawler to match a regular web browser, they are easily identifiable as an irregular
user since they do not load third-party objects and JavaScript. Such crawlers are blocked by many
websites and bot-mitigation tools [152]. In contrast, we perform complete page loads, including
third-party content and execution of JavaScript. As a consequence, our crawls are slower, requiring
around 12 hours for 500 page loads (compared to 1-2 minutes required by the urllib2 crawler).

To understand the impact of (2) and (3), we compare the discrimination results obtained from
a single front-page crawl performed by both crawlers. We started both crawls on the same day,
on the same set of websites, using the same set of 100 randomly sampled exit relays. The results,
illustrated in Figure 5.10, confirm that previous work underestimates the amount of discrimination.

![Figure 5.10: Synthetic measurements: Impact of methodological changes on measured discrimina-
tion from data generated by a single front-page crawl.](image)

### 5.5 User-driven Discrimination Measurements

While our crawls systematically explore popular websites, they might not be typical of actual Tor
usage. Thus, we performed privacy-sensitive logging on our deployed exit relays to measure how
commonly users interacting with the Alexa Top 1M web-pages experienced failed SSL handshakes
or HTTP requests. This observational dataset, based on actual Tor-user web traffic distributions
and user interactions, provides us with a more natural picture of the discrimination encountered
by actual users.
5.5.1 Logging Approach

We maintain counters for several events of interest associated with users browsing websites in the Alexa Top 1M. Our approach, designed after consultation with members of the Tor developer community, takes precautions to avoid deanonymization of users. Since neither the Tor users nor the service operators were the subjects of our study, we were exempt from an IRB review.

First, we use bucketing and split the Alexa Top 1M websites into exponentially growing sets based on their Alexa ranks as follows: The first set contains the top 100 websites (ranked 1–100) and the \( n \)th set for \( n > 1 \) contains the top \( 100 \times 2^{n-2} + 1 \) to \( 100 \times 2^{n-1} \) websites. We then maintain separate event counter for each set. Second, we maintain our event counters in memory and write to disk only once a day. This allows our event counters to attain higher count values, increasing anonymity set sizes. Third, to deal with the possibility of encountering cases where 24 hours is insufficient to achieve reasonably high anonymity set sizes — e.g., if only one person visited a site during a 24 hour period, we round-up each event counter to the nearest multiple of eight before writing to disk. A similar approach is used by Tor metrics [5] for reporting counts of bridge users per country.

We maintained per-bucket event counters for (1) the number of HTTP requests to website front pages and the number of error status codes observed in their corresponding responses, (2) the number of HTTP(S) handshakes initiated and the number of timed-out handshakes encountered. Additionally, we also maintained a counter for the number of packets sent through each open port.

5.5.2 Results

Table 5.4 shows the percentage of failed HTTP requests and incomplete HTTPS handshakes encountered by users of our exit relays. We find that the fraction of incomplete handshakes steadily increases over time. The steep increase in HTTP error codes received during weeks 4 and 5 is attributed to our relays being (ab)used in a scraping attempt on a popular website (a complaint notice was received due to this behavior). Besides this sudden increase, we see that the fraction of HTTP errors is in line with data observed through our crawls, but the fraction of incomplete HTTPS handshakes is higher. This is likely because incomplete handshakes are a very noisy indicator for user discrimination, with many reasons for them to occur naturally.

<table>
<thead>
<tr>
<th>Week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>15.8</td>
<td>18.1</td>
<td>19.8</td>
<td>32.8</td>
<td>33.4</td>
<td>17.9</td>
</tr>
<tr>
<td>HTTPS</td>
<td>36.3</td>
<td>35.0</td>
<td>41.1</td>
<td>45.2</td>
<td>47.9</td>
<td>49.6</td>
</tr>
</tbody>
</table>

Table 5.4: The percentage of failed HTTP requests and incomplete HTTPS handshakes observed over time.

**HTTP requests and error response codes.** For exiting packets using the HTTP protocol, \( \text{iff} \) the URI on the HTTP request exactly matched a Top 1M website, we incremented a front-page request event counter associated with the set containing the site. For every matching request,
5.6. CONCLUSIONS

State was maintained to identify the corresponding response packet. If the corresponding response packet contained an error status code (4XX/5XX), an error-status event counter associated with the corresponding set was incremented. We breakdown the fraction of errors by website ranks and time in Figure 5.11(a). We see that the fraction of error response codes is nearly evenly distributed across each set, indicating that errors are independent of website ranks.

Figure 5.11: User-driven measurements: Fraction of errors encountered by users visiting the Top 1M websites over time. The URL category S1 consists of the top (1-100) websites and Sn (n ≥ 2) consists of sites in the top [100 × 2^n−2 + 1 to 100 × 2^n−1] ranks.

HTTPS handshake initiation and failure. The procedure for HTTPS is similar to that for HTTP. However, we use the SNI value of client-hello handshake initiation packets instead of the URI of HTTP requests. Furthermore, we look for handshake failures and timed-outs instead of HTTP errors. The results in Figure 5.11(b) show a strong increasing trend in incompletion rates over time.

5.6 Conclusions

Limitations. Our study comes with its own limitations. Our crawls, while more in-depth than prior efforts [28], were too time consuming to run often enough to gain statistical guarantees about discrimination by any one website. Nevertheless, they show that discrimination is common and sometimes subtle.

Implications for Tor. The large amounts of blocking and discrimination identified by our crawling and privacy-preserving measurements suggest that Tor’s utility is threatened by online service providers opting to stifle Tor users’ access to their services (§5.4 & §5.5). Given the large footprints of the observed abuse, we believe future research can provide tools to curb such abuse while preserving privacy and Tor functionality. We envision Tor nodes using cryptographic protocols, such as secure multi-party computation and zero-knowledge proofs, to detect and deter users producing large amounts of traffic in patterns indicative of abuse. For example, Tor could compute privacy-preserving global counts of visits to each threatened domain and throttle exiting traffic to ones that appear over visited.

Implications for online services. Combining our study results, we can put the difficulties facing
CHAPTER 5. MEASURING THE IMPACT OF USERS ABUSING ANONYMITY

Tor users and online service operators into perspective: over 20% of the top 500 websites block Tor users. Given that Tor users do make purchases at the same rate as non-Tor users [137], this response may be excessive and operators might wish to use less restrictive means of stifling abuse.

Operators can aid Tor in developing approaches to curb abuse or unilaterally adopt local solutions. For example, instead of outright blocking, servers could rate-limit users exiting from Tor for certain webpages (e.g., login pages). Indeed, CloudFlare is developing a cryptographic scheme using blindly signed tokens to rate limit Tor users’ access to websites it hosts [153].

Ultimately, we do not view IP-based blacklisting as a suitable long-term solution to abuse. In addition to Tor aggregating together users’ reputations, IPv4 address exhaustion has resulted in IP address sharing. IPv6 may introduce the opposite problem: the abundance of addresses may make it too easy for a single user to rapidly change addresses. Thus, in the long run, we believe that online service operators should shift to more advanced ways of curbing abuse, ideally, ones compatible with Tor.
5.6. CONCLUSIONS

(a) Top 100 front-page blockers
Figure 5.12: Synthetic measurements: The websites in Alexa Top 500 found to discriminate against most of our selected Tor exit relays. Dark blue blocks indicate that every attempt to perform the interaction from the corresponding relay was discriminated against and light blue blocks indicate that at-least one of our attempts was. Figures 5.12(b) and 5.12(c) only report cases where the front-page was not discriminated against, but the interaction was.
Chapter 6

Dataset and Software Releases

In order to ensure reproducibility of our results and ease comparative evaluation, we have released the following datasets and software. For each of the following datasets and (research prototype) software, we follow the principles of the CRAPL licence[1] and take no responsibility for anything that might happen to you, your computer, or your network while running the software. We provide best-effort support for each release via the following email address: me@rishabn.com.

- **CS-BuFLO website fingerprinting defense.** We have released a fully working implementation of CS-BuFLO. This implementation, along with traces used in our experiments are available for download at https://github.com/xiang-cai/CSBuFLO.

- **Glove website fingerprinting defense.** We have released a Glove simulator along with sample network traces generated by it. This simulator is available for download at https://bitbucket.org/rishabn/glove.

- **Astoria AS-aware Tor client.** We have released a fully working implementation of the Astoria AS-aware Tor client at https://github.com/sbunrg/Astoria. Please note that the client is a research prototype and should not be considered secure or safe enough for browsing sensitive content.

- **Castle censorship circumvention framework.** We have released an implementation of the Castle framework that for legal reasons does not include game specific implementations or features (e.g., replay decoders or map generators). However, we do provide step-by-step instructions for implementing these features on generic real-time strategy games. Instructions and framework source code are available for download at https://github.com/bridgar/Castle-Covert-Channel.

- **Crawler Incantatus interactive web crawler.** We have released the source code of Crawler Incantatus at https://bitbucket.org/rishabn/crawler-incantatus.

Bibliography


[19] Roger Dingledine. Improving Tor’s anonymity by changing entry guard parameters. The Tor Blog.


[70] How the NSA Attacks Tor/Firefox Users With QUANTUM and FOXACID. https://www.schneier.com/blog/archives/2013/10/how_the_nsa_att.html


[97] CAIDA. https://www.caida.org/funding/hijacks/


[140] Mike Perry. The Trouble with CloudFlare. [https://blog.torproject.org/blog/trouble-cloudflare](https://blog.torproject.org/blog/trouble-cloudflare)


[146] Contributors to the Tor Project. List Of Services Blocking Tor. Tor project Wiki. Available at [https://trac.torproject.org/projects/tor/wiki/org/doc/ListOfServicesBlockingTor](https://trac.torproject.org/projects/tor/wiki/org/doc/ListOfServicesBlockingTor)


