TÉCNICO LISBOA

Hardening Tor against State-Level Traffic Correlation Attacks with K-Anonymous Circuits

Vítor Manuel Sobrinho Nunes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Nuno Miguel Carvalho dos Santos

Examination Committee

Chairperson: Prof. Paolo Romano
Supervisor: Prof. Nuno Miguel Carvalho dos Santos
Member of the Committee: Prof. Henrique João Lopes Domingos

January 2021

Acknowledgments

Firstly, I would like to thank my advisor, Prof. Nuno Santos, for welcoming as his student, for all the feedback and support throughout this year. Thanks for the patience, perseverance and guidance to overcome every challenge. Secondly, I would like to thank professor Luís Rodrigues for the initial ideas and discussion regarding the system design and attack defenses.

I would also express my gratitude to Diogo Barradas for being always available for fruitful discussions, ideas, feedback and for all of his support.

I want to thank my dear friends for all support and friendship during my student years and for making them an amazing journey.

Lastly and foremost I would like to thank my parents for all support and encouragement throughout my studies which would not have been possible without them.

Resumo

O Tor é uma rede sobreposta de baixa latência que permite aos seus utilizadores navegar na Internet de forma anónima. A rede Tor é inerentemente vulnerável a ataques de correlação de tráfego, que permitem a um atacante relacionar o endereço IP de origem com o respetivo endereço IP de destino. Tal pode ser orquestrado e executado por um adversário de larga escala bastando para isso, observar ambos os extremos de um circuito Tor e assim correlacionar os fluxos de saída e de entrada através de técnicas de analise de tráfego. Ainda que esta vulnerabilidade seja bem conhecida, o Tor foi considerado seguro durante bastante tempo devido à complexidade prática em concretizar o ataque. No entanto, tal complexidade tem vindo a diminuir sobretudo devido ao pequeno número de Fornecedores de Acesso à Internet, vulgarmente designados de ISPs, que expandiram rapidamente as suas infraestruturas de rede ao ponto de, atualmente, serem capazes de intercetar a maior parte do tráfego de Internet, incluindo tráfego Tor. Infelizmente, não existem defesas práticas atualmente que não requeiram alterações aos protocolos e infraestruturas do Tor. Para superar tal vulnerabilidade, nós apresentamos o TorK - um *pluggable transport* compatível com o Tor que assenta na noção de fluxos indistinguíveis para modelar o tráfego Tor de K múltiplos clientes, de forma coordenada oferecendo k-anonimato. Um adversário de larga escala que consiga intercetar a maior parte de tráfego Tor não irá contudo identificar o cliente fonte através de um circuito TorK, num conjunto de K fontes igualmente prováveis. Nós implementámos um protótipo funcional de uma ponte (*bridge*) TorK capaz de ligar clientes Tor e os seus respetivos circuitos. Através de uma análise experimental exaustiva, mostramos que o TorK é capaz de oferecer propriedades de segurança adicionais; é resiliente contra o estado da arte de ataques de correlação oferecendo, ao mesmo tempo, uma boa performance à custa de um ligeiro, mas tolerável, aumento no consumo da largura de banda.

Palavras-chave: Rede de Anonimato Tor, Ataques de Correlação de Tráfego, K-Anonimato

Abstract

Tor is a popular low-latency overlay network that allows users to surf the Internet anonymously. However, similar to other low-latency anonymity systems, Tor is inherently vulnerable to traffic correlation attacks, which enable an attacker to match a source IP with a destination IP of Tor communications. This can be accomplished by a state-level adversary capable of observing both endpoints of a Tor circuit and by correlating the corresponding outgoing and incoming flows through traffic analysis. Although this vulnerability is a well-known weakness in Tor's design, this system has nevertheless been considered secure for many years now due to the practical difficulty of mounting said attacks. However, this complexity has been steadily decreasing as a relatively small number of large ISPs has dramatically expanded their network infrastructures to the point that, today, they are able to intercept most of the Internet traffic, including Tor's. Unfortunately, no practical defenses are currently known without requiring significant changes to Tor's protocols or infrastructure. To overcome this problem, we present TorK – a Tor pluggable transport that leverages the notion of indistinguishable flows to modulate the Tor traffic of multiple cooperating K clients and provide *k-anonymity*. Put simply, a state-level adversary that can intercept the entirety of the Tor traffic, will not be able to deanonymize the real source behind a TorK-shielded circuit from amongst a set of K equally plausible Tor circuit sources. We implemented a fully-functional TorK bridge that can seamlessly interconnect Tor clients and relays. Through extensive experimental evaluation, we show that TorK offers strong security properties; it is resilient to state-of-the-art traffic correlation attacks while delivering good performance at the expense of increased (yet tolerable) bandwidth consumption.

Keywords: The Tor Anonymity Network, Traffic Correlation Attacks, K-Anonymity

Contents

Acknowledgments . . . iii
Resumo . . . v
Abstract . . . vii
List of Tables . . . xi
List of Figures . . . xiii

1 Introduction 1
1.1 Motivation . . . 1
1.2 Contributions . . . 3
1.3 Structure of the Document . . . 3

2 Background 5
2.1 The Tor Anonymity Network . . . 5
2.2 Bridges and Pluggable Transports . . . 6
2.3 Attacks Leveraging Malicious Tor Entities . . . 7
2.4 Attacks Leveraging Traffic Analysis . . . 9

3 Related Work 13
3.1 Avoiding Unsafe Relay Nodes . . . 13
3.2 Avoiding Unsafe Autonomous Systems . . . 15
3.3 K-anonymity and Existing Applications . . . 18
3.4 Discussion . . . 20

4 Design 23
4.1 System Overview . . . 23
4.2 Thwarting Protocol Level Attacks . . . 27
4.3 Thwarting Traffic Pattern Attacks . . . 32
4.4 Thwarting Bridge Impersonation Attacks . . . 34
4.5 Thwarting Client-side Sybil Attacks . . . 35

5 Implementation 37
5.1 Implementation Overview . . . 37
5.2 Pluggable Transport . . . 38

5.3 SOCKS Proxy 39
5.4 Tor Controller 40
5.5 TorK Controller Interface (CLI) 41
5.6 Traffic Shaper 41
5.7 Bridge/Client Controller 43
5.8 Shielding the TorK Bridge with SGX Enclaves 45

6 Evaluation **49**
6.1 Methodology 49
6.2 Performance 51
6.3 Resistance against Traffic Analysis 54
6.4 Resource Utilization 59
6.5 Hardening TorK against Untrusted Bridges 61

7 Conclusions **65**
7.1 Achievements 65
7.2 Future Work 66

Bibliography **67**

List of Tables

3.1 Example of K-anonymity applied to medical records. Private information (left) and public anonymized data (right). 19

6.1 Tor cells TCP length distribution. 51

List of Figures

2.1 Examples of vanilla Tor circuits. Alice constructs a circuit $(1 \leftrightarrow 2 \leftrightarrow 3)$ to access to $a.com$. Bob constructs a circuit $(1 \leftrightarrow 4 \leftrightarrow 5)$ in order to access also to $a.com$. Charlie constructs a circuit $(4 \leftrightarrow 5 \leftrightarrow 3)$ in order to access to $b.com$. 6
2.2 BGP hijacking and interception attack. 10

3.1 AS-aware path predictions predicts the n shortest paths, checking for the presence of the same AS in both paths. 16
3.2 Alibi routing as a solution to avoid unsafe geographical regions. 17

4.1 TorK architecture representing a k-circuit. All hops are observed by a state-level adversary – its domain of influence is indicated by the dashed red line. Tor onion layers are colored. 24
4.2 TorK architecture representing a k-circuit. All hops are observed by the attacker – dashed red line. Tor onion layers are colored. All three members opened a Tor circuit. 25
4.3 TorK gateways and bridge message exchange. 28
4.4 Detailed example of TorK protocol and their exchange messages. 29
4.5 TorK Protocol: Client State Machine 30
4.6 TorK Protocol: Bridge K-Circuit State Machine denoting K-Circuit status (σ). 31
4.7 Bandwidth control measure in a k-circuit with k = 2. (Only relevant chaff messages are presented - orange). 33
4.8 Scenario where the TorK bridge is controlled by an attacker. 34
4.9 Scenario where the two TorK clients are controlled by an attacker colluding to deanonymize one user. 35

5.1 Overview of TorK's components with Tor. 38
5.2 Pluggable Transport configuration and TorK Socks Proxy. 39
5.3 Tor source code change to prevent marking bridges as down upon CHANGE operations. File: `tor/src/feature/client/entrynodes.c` 39
5.4 Tor controller implementation. 40
5.5 TorK traffic shaper. 42
5.6 Traffic Shaper state machine. 43
5.7 Format of TorK frames. 44
5.8 Format of TorK control frames. 45

5.9 TorK Bridge running inside an enclave. 46

6.1 TorK throughput for the baseline (536 bytes chunk) with an increasing number of clients. . 52
6.2 Baseline configuration latency observed by number of bridge clients. 53
6.3 Observed throughput with different chunk sizes. All clients are receiving data. 54
6.4 Throughput and latency by number of clients when using different chunk sizes; clients are receiving data. 55
6.5 Performance of the XGBoost classifier when attempting to distinguish between k-circuits carrying real Tor data and chaff. 56
6.6 Performance of the XGBoost classifier when matching a given TorK flow with the corresponding Tor circuit. 57
6.7 Bytes received (1) and send (2) by each client and the bridge during a regular scenario. . 58
6.8 Bytes received (1) / sent (2) by clients & bridge when a client abruptly closes the connection. 58
6.9 Bytes received by each client when dropping all packets from client 2, without (left) and with (right) bandwidth protection. 59
6.10 Bridge CPU usage when clients receive chaff (left) and when they receive data (right). . . 60
6.11 Bridge outward (towards clients and Tor Network) network usage according chunk size/TS rate. 60
6.12 Iperf client outgoing (towards bridge) network usage according chunk size/TS rate configuration. 61
6.13 Decoy variable added to TorK. 61
6.14 TorK memory regions, whose stack lies between `0x7ffd462ea000` and `0x7ffd4630b000`. . 62
6.15 GNU Debugger (GDB) extracting a memory dump from an alive TorK bridge process. . . 62
6.16 Regular stack memory dump containing the decoy variable. 62
6.17 SGX stack memory dump. 62
6.18 Throughput of TorK-SGX baseline (536 bytes chunk) varying the number of clients. 63

Chapter 1

Introduction

This thesis addresses the problem of traffic correlation attacks on Tor and studies the use of an extension to Tor that offers additional robustness against flow correlation attempts. Providing robustness against correlation implies that an adversary that has a global view over the entire Tor network (client, relays and destination) should not, by any passive or active means, be able to find a link between clients and their destinations. However, due to several practical limitations (e.g. performance, reliability, maintainability), the problem of resisting traffic correlation attacks in Tor remains unresolved in the literature.

1.1 Motivation

Tor [1] is a widely used overlay network which allows a user to browse the Internet anonymously, i.e. without exposing its IP address to the destination. Whenever a user wants to connect to a web page, Tor starts by constructing an encrypted *circuit* based on three relay nodes – called *entry*, *middle* and *exit* relays. The communication is then forwarded along those relays which implement a variant of the onion routing protocol (that uses nested levels of encryption) to tunnel network packets in such a way that no relay can know both the source IP and the destination IP of those packets. Given these properties, Tor has been used by journalists and whistleblowers for protecting their privacy online [2] and has also been used as an Internet censorship circumvention tool to allow the access to blocked websites [2].

However, despite the sophisticated defenses of the Tor network, Tor users may be subjected to deanonymization attacks based on *traffic correlation* [3]. Such attacks can be carried out when an adversary can observe the traffic at the circuit's entry and exit relays. By intercepting network flows at these relay nodes, the adversary can use multiple flow characteristics, such as the volume of traffic or inter-packet arrival times, to perform flow correlation. If two flows are successfully correlated to one another, the adversary can determine the IP addresses of the circuit endpoints with high confidence.

Today, traffic correlation attacks are realistic threats to Tor users since they can be launched at large scale by state-level adversaries. Nithyanand et al. [4] presented an empirical study where they discovered that up to 40% of all Tor traffic is vulnerable to attacks by traffic correlation from network-level attackers; if the network-level attackers are Autonomous Systems (ASes) that may collude with each other the percentage of vulnerable Tor traffic increases up to 42%. If the same network-level attacker are state-level adversaries, then 85% Tor traffic is vulnerable to attacks. Users in some countries (China and Iran) are particularly affected, since 95% of all possible circuits are vulnerable to traffic correlation. A second study by Johnson et al. [3], shows that 80% of all users can be deanonymized when facing a relay-level attacker. Surprisingly, most of users can be successfully deanonymized within three months by an AS-level attacker. Colluding ASes can correctly correlate users in 90$\times$ less time than a single AS network-level attacker. By targeting a specific web user, colluding ASes can effectively deanonymize it in a single day.

Our goal is to build a system that erodes the power of traffic correlation attacks. Our key idea is to cast plausible doubt in the adversary with regards to the real originator of a specific circuit (i.e., the source). In particular, we propose the use of k indistinguishable flows, such that the exit flow of a given circuit can no longer be correlated with a single client ingress flow. Instead, it would be correlated with a set of k incoming client flows, where any of the flows can be the actual source of the exit flow. Essentially, this provides *k-anonymity* [5] for a particular Tor circuit: the adversary will not be able to distinguish the original sender's identity amongst a set of k possible other senders, thereby weakening its ability to precisely identify the actual sender.

Our approach is based on the idea of constructing k*-circuits*: Tor circuits with k ingress flows and multiple egress flows, such that is not possible to link an ingress to a corresponding egress flow. To build k-circuits in the Tor network we must overcome several challenges. First, Tor circuits currently offer point-to-point channels. However, in order to enable k cooperating users to participate in the creation of a k-circuit (to help a given user to get in touch with a certain destination) we need to perform several adaptations in the way Tor circuits are built. In particular, we require the generation of k ingress flows that can be coalesced into a single flow. Ideally, we wish to provide this service without changing the underlying Tor protocols or the existing software implementations. Second, we must ensure that the k ingress flows cannot be distinguished by the adversary, even when employing advanced traffic classifiers; i.e., the k ingress flows must be *indistinguishable*. Third, the performance overheads introduced by our components should be tolerable to the end users.

In this work, we propose to build TorK, a system that aims to provide k-circuits for unmodified Tor infrastructure. To build TorK, we plan to combine several ideas. First, we leverage the concept of Tor *bridges* to create trusted nodes for setting up k groups. Users willing to establish a k-circuit connect to a TorK bridge, special nodes that shape all Tor traffic between the client and its first hop [6], which will instruct the sender to wait until multiple other $k-1$ members contact the bridge. Once the number k of pending connections has been satisfied, TorK creates and gathers clients in a group of k clients – *k-circuit*. TorK employs a synchronization protocol among members of k-circuits so that members can join efforts and act together to achieve robustness to correlation attacks. To prevent the adversary from distinguishing traffic patterns between each of the k members and the TorK bridge – which acts like a hub – the traffic between the bridge and each k-member will be morphed [7] in order to guarantee the indistinguishability of client-generated flows.

1.2 Contributions

This thesis analyzes, implements and evaluates a practical solution to correlation attacks on Tor by providing additional properties such as k-anonymity to the current Tor infrastructure. In particular, this thesis makes the following contributions:

- Proposes an extension to the current Tor infrastructure, named TorK, which enforces the creation of indistinguishable channels among a pre-defined number of clients. This channel allows for the tunneling of Tor cells. TorK features a synchronization mechanism which is responsible for gathering clients in a k-anonymity group.
- Evaluates the system under different configuration trade-offs regarding throughput, latency, robustness against active and passive attacks, CPU, and network usage.
- Studies the above trade-offs to determine the optimal TorK configuration parameters.
- Evaluates the robustness of the system considering local adversaries, i.e. adversaries possessing the ability to launch malicious relays or otherwise compromise live Tor relays in order to access TorK memory space.

1.3 Structure of the Document

The remaining of this document is organized as follows. Chapter 2 provides an introduction of correlation attacks as well as a description of different types of attacks targeting anonymous networks, in particular Tor. Chapter 3 provides a comprehensive detail and discussion about literature solution to mitigate Tor correlation attacks. Chapter 4 describes the architecture of TorK. Chapter 5 describes implementation details of TorK's prototype. Chapter 6 presents the results of the experimental evaluation. Chapter 7 concludes this document stressing the main findings and pointing directions for future work.

Chapter 2

Background

In this chapter, we start by providing an overview of the current Tor system, and explain how it provides sender and receiver anonymity. We discuss with more detail some Tor mechanisms that we leverage for building TorK, namely bridges and pluggable transports. Then, we identify the main traffic correlation attacks to the Tor network that can be launched by a state-level adversary.

2.1 The Tor Anonymity Network

Tor is a circuit-based low-latency anonymous communication network [1] which implements a variant of *onion routing*. The onion routing protocol implemented by Tor aims to provide sender anonymity for TCP-based applications such as web browsing. Tor relies on nodes – designated by *relays* or Onion Routers (OR) – which are maintained by volunteers and forward traffic along a circuit (see Figure 2.1). Circuits are essentially composed of three relays: an *entry*, a *middle* and an *exit* relay (or node). Clients choose relays when constructing circuits by selecting them from an available list. This list is available at special relays that act as *directory authorities*. In a circuit, onion router relays only know their predecessor and successor and no other relays. Data flows through circuits in fixed-size *cells* (512 bytes) encrypted by symmetric keys previously exchanged with clients. Each cell is intended to a specific relay. Cells can be of two types: *relay* and *control* cells. Relay cells contain end-to-end data and control cells are interpreted by the OR that receives them (e.g. extend circuits). Relays can multiplex multiple TCP streams [1] along each circuit to improve efficiency and anonymity. Relay cells also contained end-to-end integrity checking checksums allowing last relays to identify defective cells and discard them.

Tor uses a small group of relays as directory authorities. They maintain a signed document of the current Tor network status which includes relays list, their certificates and public keys. Clients can fetch this document from any directory authority to obtain a list of available relays. Later, Tor added directory caches which obtain a copy from directory authorities [8]. Other authorities or clients fetch directories from caches, doling out bandwidth. Periodically, onion routers sign and publish their router descriptors [8], containing their keys, capabilities, and other optional information. Directory authorities gather the router descriptors and generate a signed view of the network. They send a summary of that view to

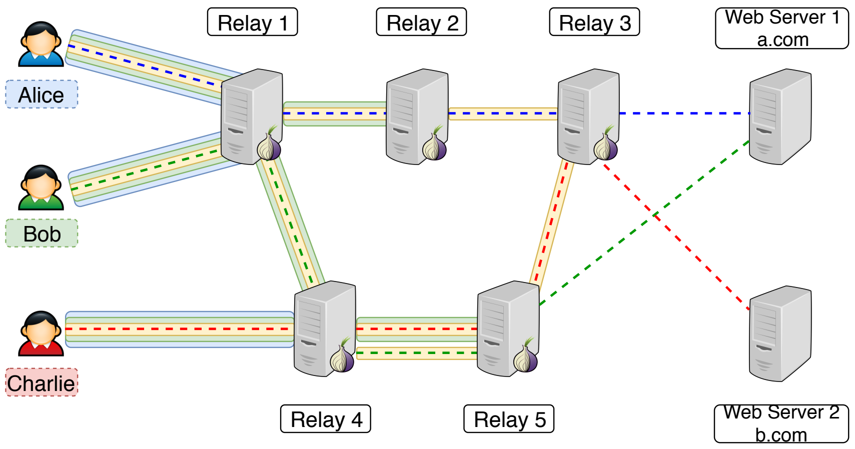


Figure 2.1: Examples of vanilla Tor circuits. Alice constructs a circuit $(1 \leftrightarrow 2 \leftrightarrow 3)$ to access to $a.com$. Bob constructs a circuit $(1 \leftrightarrow 4 \leftrightarrow 5)$ in order to access also to $a.com$. Charlie constructs a circuit $(4 \leftrightarrow 5 \leftrightarrow 3)$ in order to access to $b.com$.

other directory authorities. All summaries are then voted to build a signed document – designated as *consensus* document – that describes the current status of the Tor network [8].

To establish a circuit, clients obtain a list of current relays, and then exchange session symmetric keys with each OR in a circuit, one at a time – referred to by *telescoping* path-build design [1]. These symmetric keys are valid during the session. ORs discards keys when the circuit is closed providing perfect forward secrecy. Tor is compatible with the majority of TCP-applications without any modifications or kernel requirements. By providing a SOCKS interface, it allows, e.g., an email client to be used on top of Tor without any modifications on the email client itself apart from changing a proxy configuration.

2.2 Bridges and Pluggable Transports

Motivated by the anonymity properties offered by Tor, many users adopt this system as an Internet circumvention tool within censored regimes to access blocked web sites. For this reason, Tor has become a major target for blocking by the ISPs of such regimes. One way for blocking Tor traffic is by blacklisting Tor relays. Since the relay list is public it is trivial to blacklist the IP addresses of all Tor relays, preventing Tor clients from establishing circuits. To circumvent this attack, Tor employs *bridges*. Bridges consist of unlisted proxies whose goal is to forward client's traffic to an entry relay. Thus, rather than connecting to some potentially blacklisted entry relay, clients connect instead to some unlisted bridge (hopefully) unknown by the adversary. Some bridges are *public*, others *private*. Public bridges are deployed by volunteers to be used by any Tor user. Private bridges are exclusively for who knows about their existence [9].

However, bridges are still vulnerable to traffic fingerprinting attacks. If a user connects to a bridge using the vanilla Tor protocol, it becomes possible to identify Tor traffic and blacklist that bridge. Matic et al. [9] show that 55% of public bridges are vulnerable to aggressive blocking. This reason is that Tor traffic has unique properties that greatly facilitate the job of ISPs in identifying and block it. First, Tor uses fixed-size 512 byte cells which results in a characteristic packet size distribution that can be analyzed by an adversary in order to identify with high probability if a given flow carries Tor cells or not. Second, Tor traffic can be detected based on TLS byte patterns. Tor communications (e.g., between relays) are encrypted using TLS. Server Name Indication (SNI) is a TLS extension [10] that adds the domain name to the TLS header in order to be used in the *Client Hello* handshake phase. However, the SNI is not encrypted – since it occurs before the TLS handshake – allowing an eavesdropper to discover the domain name that the client is trying to reach. Tor uses a fixed random string as a bogus domain name (e.g. www.kf3iammnyp.com) and a specific cipher suite set [11]. These properties uniquely identify encrypted traffic as Tor's, enabling censors to block new bridges minutes after they have been deployed by simply observing these two properties.

To surpass identification, Tor bridges support *pluggable transports* (PT) [6] which are obfuscation wrappers that shield the Tor traffic between a client and a bridge from traffic analysis. The most popular PTs are *meek* [12] and obfs4 [13], which is based on obfs2 [14] and provides a level of obfuscation for an existing authenticated protocol, like SSH or TLS. The obfs4 protocol encrypts all traffic using a symmetric key shared between the bridge and the user derived from both client's and bridge's initial key [11]. An attacker would see a randomly encrypted byte stream. *meek* uses a technique called *Domain Fronting* [12]. This technique consists on using different domain names in the SNI and in the HTTP *Host*. The idea is to encode a legitimate website domain name into the SNI (e.g. the address of a public cloud) and use the Host field in the encrypted HTTP request to ask for access to the forbidden one. *meek* requires a CDN (Content Delivery Network) like public cloud providers which supports domain fronting. Bridges can support multiple PTs in order to serve a larger number of clients. However, bridges that offer more than one PT may reduce the security of the most secure PTs they offer. Similarly, bridges running non-Tor services (e.g. SSH) also reduce the level of security provided to users [9].

2.3 Attacks Leveraging Malicious Tor Entities

One of the most powerful classes of attacks aimed at deanonymizing Tor circuits are the so called *correlation attacks*. A correlation attack is an end-to-end attack where an adversary searches for a correlation between two flows on a given entry and an exit relay. Since an entry relay connects directly with a client and an exit relay connects directly with a server, an attacker can conclude that a client is accessing a certain server if it finds that two observed flows are correlated. State-level adversaries find themselves in a particularly good standing for being able to launch such attacks due to their privileged control over a country's network. We briefly survey some of the most effective correlation attacks to Tor known today that rely on the existence of malicious entities in control of the adversary, such as clients, relays or directory authorities.

2.3.1 Timing Attacks

These are end-to-end active attacks where an attacker attempts to determine the endpoints of a circuit by correlating the time it takes for a flow to travel from the entry to the exit relay. Chakravarty et al. [15]

employ single-end bandwidth estimation for deanonymizing the source of a given Tor connection. The attacker places probe servers alongside the egress and ingress routers at the boundary of ASes. Assuming that the client will connect to a server colluding with the adversary, the server will then send traffic along with a pattern that can be detected by the attacker's probing nodes. In this way, the attacker can discover the AS of the client and may further escalate the attack in order to pinpoint the actual client's location.

Pries et al. [16] presented a different timing attack based on the disruption of the AES-CTR counter used to encrypt Tor cells. This attack requires the entry and exit node to be controlled by an attacker and works by duplicating a cell in the entry node, which causes a decryption error at the exit node. An attacker can correlate the time of the cell transmission with the time of the decryption error to ensure that the error was caused by the cell duplication and ultimately identify the source of the connection.

Murdoch et al. [17] propose some defensive strategies in order to mitigate timing attacks. One strategy designated by *perfect interference* state that the output streams should all have the same shape, e.g., using threshold mix batching. However, Tor does not employ threshold mix batching which relies on waiting for several streams and then flush them, all at once. Such technique would increase the latency defeating the low-latency characteristic of Tor. To prevent timing attacks, Tor relays send cells from different streams in a round-robin fashion.

2.3.2 Sybil Attacks

These attacks have a supportive intent for other kinds of attacks launched against the Tor network. In particular, Sybil attacks consist on the deployment of multiple malicious relays controlled by an attacker, while creating the illusion of pertaining to different identities. The idea is to obtain a large disproportional network influence [18] which goal is to divert user traffic to malicious relays. In fact, a wealth of attacks on Tor, such as correlation attacks or timing attacks, depend on the amount of network traffic the attacker can control and on the ability to trick users into connecting through malicious relays [19].

Defenses against Sybil attacks are not trivial since it is always possible to perform them unless a central authority is used to assure a correspondence between a relay and an identity [20]. However, using a central authority contradicts Tor's goal that eliminate single points of control. Such central authority would increase barriers on new relays' deployment, requiring relay operators attestation by central authorities. Another approach proposed by Bauer et al. [21] consists in limiting the number of new accepted relays at directory authorities. The idea is to limit the number of relays by IP address subnet. An attacker would need to have access to multiple subnets to conduct the attack. Both solutions do not directly stop sybil attacks but increase the attacker's startup cost.

2.3.3 Predecessor Attacks

Repeated communication between two endpoints of a Tor circuit may open the door to vulnerabilities that can be exploited by a traffic analysis-capable adversary [22]. In order to perform a predecessor attack, an attacker must compromise an entry and an exit relay. This setup can arguably be made simpler through a Sybil attack, where an attacker can reduce the time to achieve control of more nodes. The goal of this attack is to learn the identity of a single or multiple senders when they are connected to a destination over time. The attacker maintains a shared counter (one per honest node) in each malicious node, initially zero. When a malicious node is selected to be a part of an anonymous circuit with a given destination, the counter of its predecessor is incremented. This counter represents the number of times a honest node is predecessor of a malicious node on circuits with the same destination. Figueiredo et al. [22] characterize the attack's capabilities, the sufficient and necessary conditions to the attacker succeed. They consider two situations regarding initiators: a single and multi initiators, both continuously send traffic to the same destination. The attacker can use the counter values to discover the set of initiators of a given circuits towards a specific destination.

Wright et al. [23] present a defense against predecessor attacks assuming a static network model, i.e. nodes do not leave the network. In this model they assumed a set of N nodes from which $C < N$ are controlled by an attacker. All N nodes communicate with the next node – the *responder* – which does not belong to N. The defense acts in rounds. In each round, each node computes a path containing all other nodes between them and the responder. In order to defend against predecessor attacks, the authors study the concept of fixing nodes in certain positions. There are variants of these defenses according to the selected positions (e.g. first, last or first and last). Considering the case of selecting the first position on the path, unless the fixed node belongs to the set of nodes controlled by the attacker, users are protected since the initiator of users' communications is indeed the node. Tor uses the same approach by imposing guard rotation restrictions [24], as further discussed in Section 3.1.

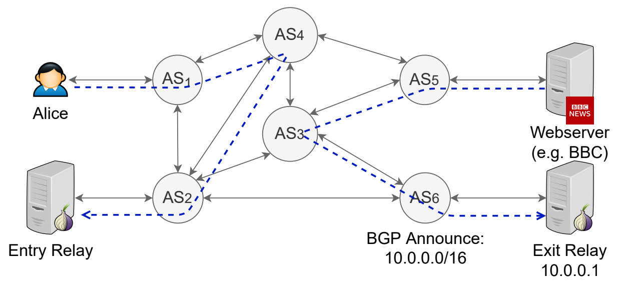
2.4 Attacks Leveraging Traffic Analysis

We explain several attacks that do not depend on the ability of the adversary to compromise Tor entities, but are leveraged by traffic analysis techniques – mostly in a passive fashion – observing the traffic in specific vantage points.

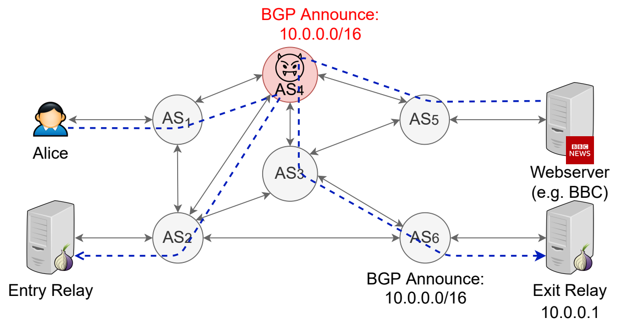
2.4.1 Circuit Fingerprinting Attack

Sun et al. [25] present an approach based on asymmetric traffic analysis allowing an adversary to correlate and deanonymize a circuit's endpoints, even if the attacker has access to different directions of the flow, e.g., from client to entry and server to exit. The rationale of this technique is based on the fact that the Internet relies on asymmetric connections, i.e. the path from the client to the server may differ from the same server back to the client, and that an adversary is still able to perform traffic correlation by observing different directions of a given flow. To augment the possibility of an adversary to observe a flow, authors additionally propose a BGP interception attack which can be launched by a malicious AS in order to divert traffic, enable traffic analysis, and forward traffic to the original destination. The following example illustrates how such an attack can be launched.

Figure 2.2(a) depicts the AS path routing between Alice (a regular Tor user) and their destination



(a) Regular scenario.



(b) AS4 hijacks and intercepts `10.0.0.0/16` subnet traffic.

Figure 2.2: BGP hijacking and interception attack.

over the Tor Network. Alice's Tor traffic travels with three different AS (AS1, AS4 and AS2) towards her entry relay. Similarly, her exit relay is connected to her destination (e.g. a news website) through three AS also (AS6, AS3, AS5). In fact, no single AS can perform correlation, since it needs to have access of both flows. If multiple AS collude a coordinated efforts, then Alice is vulnerable to a correlation attack. However, malicious AS can hijack BGP subnet addresses and consequently intercept traffic. Figure 2.2(b) displays a scenario where a malicious AS (AS4) advertises the same subnet belonging to AS6. A single AS, AS4 is going to have access to all subnet traffic, therefore, having access to simultaneously Alice entry flows (towards the entry relay) and exit flows (towards her destination).

2.4.2 Correlation-based Analysis

One of the first correlation attacks based on timing analysis was proposed by Shmatikov and Wang [26]. Inter-packet timing information is usually not carefully protected in mix networks since it would require to delay packets to hide timing patterns. An attacker can exploit this timing property by correlating the inter-packet time on both endpoint links, concluding that those links belong to the same circuit, which would tie a source to the corresponding destination. They analyzed the resilience of low-latency anonymous systems regarding correlation attacks based on packet interval time. They conduct several experiments using HTTP traces. The attacker starts by observing a set of entry and exit relays in order to link entries to exit nodes. Using an observation time of 60 seconds, the attacker divides the time into a fixed size window. An attacker counts, during each time window, the number of observed packets. The correlation between any two sequences is then computed using a cross-correlation metric. If the metric extends some threshold, the attacker can infer that both samples belong to the same flow.

The state-of-the-art work for performing flow correlation is DeepCorr [27]. DeepCorr uses advanced machine learning algorithms, instead of statistical metrics to conduct accurate flow correlation on Tor. Congestion may cause networks to suffer from bandwidth reductions, thus leading Tor packets to experience fragmentation hence decreasing the effectiveness of existent statistical flow correlation techniques. Contrary to previous attacks, DeepCorr learns a correlation function which is able to link flow samples regardless of their destination, while accounting for the unpredictability of the Tor network.

In their work, Shmatikov and Wang [26] propose an approach consisting of using intermediate relays to inject dummy packets to normalize statistical information, named *adaptive padding*. This padding would reduce the ability of an adversary to fingerprinting packets on a circuit. To protect against traffic correlation attacks based on machine learning techniques, as in DeepCorr, Nasr et al. [27] propose that Tor should enforce the use of pluggable transports across all relays, instead of just on bridges as in vanilla Tor. However, while the use of pluggable transports enables the obfuscation of both traffic patterns and content, the deployment of such a solution translates in significant performance reductions and is thus disregarded as a feasible defense against correlation attacks to be implemented in Tor. Other recent and promising countermeasures against traffic correlation attacks rely on employing AS-aware relay selection mechanisms [4, 28, 29] that effectively decrease the probability of an adversary to be in the necessary position to observe traffic and perform a correlation attack.

Summary

This chapter introduced the notion of the Tor anonymity network and their main entities. As it can be observed there are different categories of attacks to the Tor network and they can be categorized in two distinct groups: using malicious Tor entities and using traffic analysis.

Correlation-based attacks do not depend on the attackers' ability to compromise entities and therefore are much harder to detect and mitigate.

Chapter 3

Related Work

In this chapter, we introduce the main known defense strategies against traffic correlation attacks, and discuss their strengths and limitations when facing a state-level adversary. Then we provide an overview of k-anonymity, a notion of anonymity that has been employed in different domains. We propose to adopt it to weaken the ability of state-level adversaries to deanonymize Tor traffic.

3.1 Avoiding Unsafe Relay Nodes

As discussed above, by controlling the entry and exit nodes of existing Tor circuits, an adversary may be able to deanonymize them, i.e., identify the IP addresses of the corresponding sender and receiver, through multiple techniques. To mitigate adversary's attempts to launch such attacks, clients may attempt to avoid unsafe relay nodes by employing several strategies which we describe next.

3.1.1 Run a Co-located Trusted Relay Node

Instead of connecting directly to an entry node that could be controlled by an adversary, the user may run a local (trusted) relay node alongside the Tor client, and use that relay as entry node to the Tor circuits created by the user. As a result, the downstream relay nodes will not be able to determine whether the traffic forwarded by the trusted relay node was originally produced by the local user or by another user that may be using that same relay node for building its own circuits.

However, requiring every single user to run a relay node cannot be broadly implemented, partly due to the hardships in configuring the system, but mostly because many Tor clients are located behind restrictive firewalls where they cannot relay traffic [30].

3.1.2 Scan and Flag Bad Relay Nodes

Another approach is to decrease the possibility of clients selecting malicious relays, by detecting and reporting bad relay nodes. Generally, a relay is considered to be bad if it is malicious, misconfigured, or unreliable. To mark relays, Tor uses a set of labels designated as *flags* [31]. Directory authorities vote for applying those flags by measuring the bandwidth, uptime, and reliability. This allows client to select relays according its position on the circuit. For example, an entry relay should be long lived. Otherwise an attacker can setup a malicious relay and start discover client identities right away. Tor maintainers run a service aimed at verifying the reports of possibly unsafe relays [32]. They will then attempt to reproduce the problem, and possibly try to get in touch with the relay operator. If the problem cannot be solved, Tor maintainers will assign a flag – e.g., BadExit – to the reported relay thereby instructing the clients to not use it any further in the future as exit node. Tor maintainers scan the network seeking for bad relays, especially bad exit nodes, using a tool named *exitmap* [33].

Additional tools can be used for that purpose by the community in general, such as *torscanner* [1], and *tortunnel* [2]. Some of these tools use decoy traffic in order to detect bad relays [34]. Tor also imposes that relays' bandwidths should be continuously monitored by directory authorities in order to prevent malicious relays from lying about their bandwidth (for load balancing purposes, clients choose relays proportionally to their measured bandwidth capacity). Directory authorities compute weights associated with each relay class (entry, middle, exit) based on the current bandwidth of relays. This prevents a malicious relay from advertising a (fake) 100 Gbps bandwidth that would make it eligible to be selected by numerous clients.

3.1.3 Restrict the Set of Entry Nodes Used by a Client

Suppose that an adversary controls, or observes, C relays. Assume that the total number of relays in the Tor network is N. If every time a client uses the network selects a new entry and exit relays, this means the adversary can correlate the traffic sent by the client with a probability of about $(C/N)^2$, i.e., (C/N) chance of connecting to the first relay and $(C-1/N-1)$ chance for the last relay. Thus, selecting many random entry and exit nodes will leave the user in a situation where his communications could be profiled by such an adversary. As a defense mechanism against such attacks, Tor employs *entry guards* [24]: each client chooses a (guard) relay from a list of three relays when making the first hop of circuits, and uses only those relays as entry nodes. If the adversary does not control (or cannot observe) the guard, then the user is secure. Otherwise, the adversary can indeed see a larger fraction of the user's traffic, but the chance of avoiding profiling drops significantly to a probability of about $(N-C)/N$.

In the past, clients chose an entry (guard) relay from a list of three relays and each guard was discarded after a 30-60 days period. This rotation of guards helps prevent known attacks and allow to distribute client's load across multiple guards. In particular, if a client was unlucky and selected a malicious guard, he had the chance of regaining anonymity when its current guard changed. However, such parameters were not strong enough to hold a large AS-level adversary; this is described as Guard Rotation Weakness [24]. Elahi et al. [35] empirically demonstrated that Tor's time based guard rotation criteria led to clients switching guard relays more often then they should, increasing the possibility of profiling attacks. The authors mention two main threats: when a guard replaces another guard due to unavailability, and when the guard rotation occurs after a period of 30-60 days. In the first scenario,

[1] `https://code.google.com/archive/p/torscanner/` Accessed: 2020-01-05
[2] `https://moxie.org/software/tortunnel/` Accessed: 2020-01-05

selecting a malicious relay to replace an unavailable one does not present an immediate threat. The malicious relay is placed at the end of a list comprised by three guards that can be selected by the client. If the unavailable guard becomes available again, then the malicious relay is discarded and replaced with the previous guard. Selecting a malicious relay in the second scenario is a more critical threat, since the malicious guard relay will be used multiple times by the client. To achieve a better trade-off between anonymity and load balancing, clients are currently recommended to keep the same guard relay for a period of nine months. For being suitable to be elected as a guard, a relay must meet a minimal bandwidth threshold, its uptime must be greater than the median over all relays, and the node must have been present in the network consensus for at least two weeks.

3.2 Avoiding Unsafe Autonomous Systems

Unfortunately, the techniques presented above may not be effective against an adversary that can observe large fractions of the network. Such an adversary may not even need to control any specific relay node to deanonymize Tor traffic, but only to possess the ability to eavesdrop on the inter-relay traffic that crosses the network controlled by the adversary [3, 36].

To cope with this problem, others have proposed new defensive approaches against AS-level adversaries, i.e., those with the ability to access the network infrastructure of an entire Autonomous System (AS). Typically, such approaches attempt to help Tor clients' to choose paths away from the prying eyes of ASes by leveraging the analysis of the Internet topology boundaries and inter-relay latencies [37].

3.2.1 AS-level Monitoring and Tuning

As a way to prevent attacks based on traffic interception through BGP hijacking (see Section 2.4.1), Sun et al. [25] have proposed a monitoring framework for detecting BGP changes. The use of such framework allows Tor to inform vulnerable clients, which in turn can opt to suspend Tor communications or use another relay. The BGP monitoring framework uses two main heuristics. First, if some AS announces a path to a prefix it does not own frequently, the framework would alert for a possible hijacking attack. Second, if the prefix is advertised only during a small amount of time, then it might be a routing attack. In order to have a robust solution it is necessary to accurately know which ASes are traversed by a given circuit path. To this end, the proposed framework computes the traceroute of every Tor relay daily. Based on these results, it is possible to observe AS-level path changes and detect suspicious ASes.

The same authors also propose several techniques to reduce the probability of AS-level attackers to perform BGP hijacking and interception attacks. As a matter of fact, most Tor relays' IP addresses (90%) have a prefix shorter than $/24$. Therefore, authors suggest that Tor relays should not operate with prefix shorter than $/24$ prefix. However, even with a $/24$ prefix, an attacker can advertise another equal prefix. To cope with this situation, clients should prefer guard relays whose AS-level path is the shortest. Given that a smaller number of ASes in the path translates to a lower chance of one of them advertise wrong prefixes, this measure reduces the chance of a malicious AS falsely advertise routing prefixes.

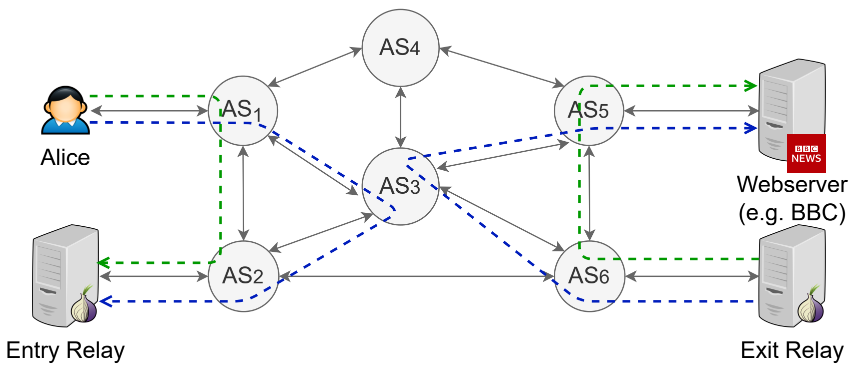


Figure 3.1: AS-aware path predictions predicts the n shortest paths, checking for the presence of the same AS in both paths.

3.2.2 AS-aware Path Prediction

In order to mitigate correlation attacks, a number of authors have proposed AS-aware path selection algorithms for decreasing the chance of an AS-level attacker to observe traffic flowing between both endpoints of a Tor circuit [4, 25, 28, 29].

Edman and Syverson [29] have experimented adding two different requirements to Tor's path selection algorithm. The first requirement mandates that each node in a circuit must be located in a different country, while the second dictates that, instead of requiring unique countries, each node should be located in a different AS. While the two approaches decreased the probability that an AS would be able to eavesdrop in both ends of a connection, they did not sufficiently mitigate the possibility for a malicious AS to perform traffic correlation. Thus, authors have proposed a more effective heuristic for safe AS path-aware selection. First, the client finds all shortest forward and reverse AS paths from the client's AS to the entry node, and from the exit node to the destination. Entry and exit paths are sorted according to the cumulative frequency values of each edge in the path. The shortest *n* paths with the greatest cumulative edge frequency values are then assumed to be the *n* most likely AS-level paths from a source AS to the destination. If the same AS appears in any of the *n* entry paths and any of the *n* exit paths, the chosen entry-exit node pair is discarded and a new pair is selected. Figure 3.1 depicts the two Alice's egress and igress Tor circuits shortest paths. The second path – marked as blue – causes this relays to be discarded by Alice since there is one path that contains the same AS in both flows.

Akhoondi et al. [28] propose LASTor, an AS-aware Tor client that selects safe paths between a client and a destination. The key insight of this technique holds on the prediction of the set of ASes through which traffic may be routed between a pair of IP addresses, instead of performing a prediction for a precise route between these addresses. The potential for the existence of a malicious AS able to perform traffic correlation is determined by checking if the intersection between the AS sets for the paths between the client and the entry relay and between the exit relay and the destination is non-empty.

Sun et al. [25] approach consists in monitoring and storing paths between the client and the guard relay, and also between the exit relay and the destinations. The idea is that each relay publishes, as part of the Tor consensus document, the list of ASes it uses to reach a given relay or destination. Clients can then use this information along their own measurements when constructing circuits. This allows clients

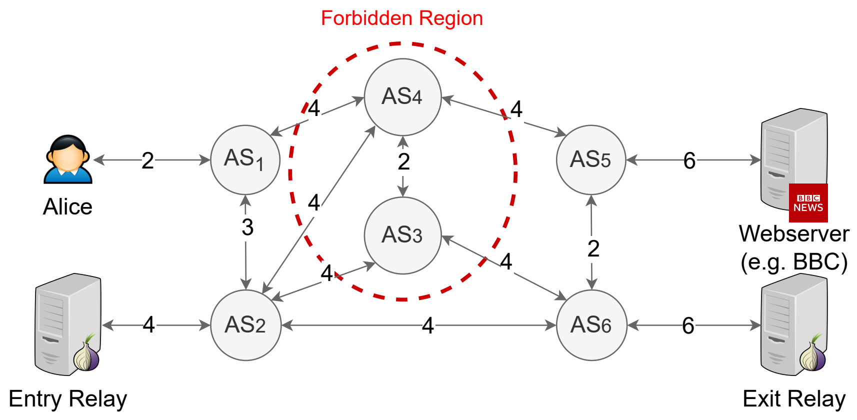


Figure 3.2: Alibi routing as a solution to avoid unsafe geographical regions.

to select relays in a way that one AS will not appear on both the entry and the exit relay.

Nithyanand et al. [4] further developed an AS-aware variant of Tor, called Astoria. Astoria avoids vulnerable circuits while employing an efficient network management by load balancing circuits across secure paths. First, Astoria takes advantage of Internet topology maps to predict which ASes are placed in critical locations for performing traffic correlation. Second, it identifies which ASes are more likely to collude with each other and increase the chance of malicious ASes to launch a successful attack. In their study, Nithyanand et al. reveal that almost half of Tor circuits (40%) constructed by current Tor clients are vulnerable to network level adversaries. This is critical in some countries [4], where state level attackers are in a position to launch attacks up to 85% of all Tor circuits. Astoria leverages a probabilistic model to select circuits on paths less liable to be correlated by an adversary, when no completely safe circuits are found. This model minimizes the amount of traffic that is observable by an attacker over time. When there is more than one safe path available Astoria causes a load balancing technique to prevent relay overload. By leveraging these techniques, Astoria is able to reduce the probability of selecting a vulnerable circuit from 40% to 3%. The path selection algorithm of Astoria is, however, arguably incomplete. Due to the existence of active BGP interception attacks [25], Astoria's Internet topology maps may become outdated for short periods of time, allowing an attacker to still launch a successful correlation attack.

3.2.3 Avoiding Unsafe Geographical Regions

Other than avoiding specific ASes, related literature focuses on avoiding entire geographic regions altogether, typically at the country-level granularity. The motivation is oftentimes the need to evade censorship policies against Tor traffic implemented by repressive governments.

Tor allows users to select a set of countries to exclude from circuit selection [30] i.e. regions to where it should not forward client's traffic. DeTor [38] presents techniques to prove that a Tor circuit did not travel within excluded regions. To provide the so called *provable geographic locations*, DeTor authors borrow the idea of *alibi routing* [39] into Tor. Alibi routing (AR) uses the packets' round trip time (RTT) and the speed of the light as a constant to prove that a given packet did not travel within forbidden regions. AR uses a single relay located outside the forbidden region to confirm that traffic is going from that relay to the destination. That confirmation is based on a message authentication code issued by the relay itself, attesting that the traffic was forwarded by that relay. If the RTT from the relay to the destination is less than the smallest RTT that also include a forbidden host, then it is possible to conclude that the packet could not travel from within the forbidden region. While AR uses one single relay, DeTor [38] generalizes this approach for three relays. Figure 3.2 exemplifies the alibi routing strategy used by DeTor. In order to avoid traffic to travel within forbidden regions, specified by Alice. If exit relay can attest that the RTT is less than the smallest RTT including a forbidden region AS, thus the package could not travel from within the AS. Namely, if the RTT of a package sent by Alice is no greater than 14 ms, then it is possible to conclude that the package could not travelled within AS3 and AS4 located under the forbidden region.

However, there are a few limitations regarding DeTor which make it an unconvincing solution for providing provable geographical avoidance. A first limitation concerns the fact that DeTor obtains the list of relays from the Tor's public database which contains several bits of information, such as: IP address, port, public key and country. If the information about the country is unavailable, DeTor uses IP geolocation services to find the exact location of Tor relays, but without any further confirmation. This may hamper the ability of DeTor to perform accurate measurements [40, 41]. A second limitation involves handling links with high latency, where it is not possible to measure the packets' travel times accurately solely relying on RTT measurements. Another limitation is that DeTor assumes a symmetric routing nature, i.e. assumes that the request and reply will traverse the same geographical locations, something that may be unlikely to happen in practice.

A more recent piece of work by Kohls et al. [41] introduces the concept of *empirical avoidance* and proposes new improvements to overcome DeTor's main limitations. In their work, authors propose TrilateraTor, a system introducing a new measurement technique that derives a circuit end-to-end timing directly from the handshake in Tor's circuit establishment procedure. To prevent the use of fake GeoIP information in its measurements, TrilateraTor leverages a distributed measurement infrastructure so as to perform trilateration and obtain accurate estimates of the physical location of Tor nodes.

3.3 K-anonymity and Existing Applications

Unfortunately, the defenses described above are not effective against a sufficiently powerful adversary that possesses the ability to observe the entire Tor network or a substantial fraction of it. In fact, these techniques are helpful insofar as they can help prevent an adversary from being able to observe entry and exit nodes of Tor circuits. However, failure in avoiding such nodes means that Tor users are potentially vulnerable to deanonymization. The following quote by the Tor maintainers warn about this important limitation (our emphasis):

> "Tor (like all current practical low-latency anonymity designs) fails when the attacker can see both ends of the communications channel. For example, suppose the attacker controls or watches the Tor relay you choose to enter the network, and also controls or watches the website you visit. In this case, *the research community knows no practical low-latency design that can reliably stop the attacker from*

Name	Age	Gender	Condition	Name	Age	Gender	Condition
Anna	30	F	Hypertension	****	30-45	*	Hypertension
Emily	23	F	Chronic Lung Disease	****	18-29	*	Chronic Lung Disease
Melissa	43	F	Diabetes	****	30-45	*	Diabetes
Edward	21	M	Diabetes	****	18-29	*	Diabetes
Ryan	29	M	Hypertension	****	18-29	*	Hypertension
Karen	49	F	Hypertension	****	45-65	F	Hypertension
Richard	88	M	High cholesterol	****	85+	M	High cholesterol
Catherine	58	F	Diabetes	****	45-65	F	Diabetes
Frank	93	M	High cholesterol	****	85+	M	High cholesterol
Adam	41	M	Chronic Lung Disease	****	30-45	*	Chronic Lung Disease

Table 3.1: Example of K-anonymity applied to medical records. Private information (left) and public anonymized data (right).

correlating volume and timing information on the two sides"[42].

Evidence suggests that such powerful adversaries exist today in the form of actors being able to control entire networks within national boundaries (oppressive regimes) [43], or to able tap into the Internet backbone across countries (intelligence agencies) [44]. To counter such attacks, we propose an alternative approach. Assuming that a state-level omniscient adversary can observe the entire Tor network, our idea is to ensure that deanonymizing a given circuit through flow correlation will not give away the identity of the original sender unequivocally, but only as one potential candidate out of K plausible users that could be responsible for the communication. In other words, we plan to enhance Tor circuits so as to provide *K-anonymity*. In this section, we provide an overview of K-anonymity and how it has been applied in multiple application domains.

3.3.1 K-anonymity in Databases

Proposed initially by Samarati and Sweeney [5], K-anonymity was introduced to solve the problem of disclosing privacy-sensitive database records that needed to be anonymized. Their original problem was formulated as follows [5]: "Given person-specific field-structured data, how to produce a release of the data with scientific guarantees that the individuals who are the subjects of the data cannot be re-identified while the data remain practically useful." K-anonymity is then a property of released data such that the information about any given individual cannot be distinguished from at least $K-1$ individuals. It follows that for a larger value of k, the anonymity set is larger, which means that anonymity is stronger. Since it has been proposed in 1998, K-anonymity has been extensively studied, e.g., in order to develop methods for k-anonymization of datasets [45], craft new attacks to k-anonymized databases [46], and propose alternative anonymity definitions [47].

Table 3.1 displays private medical records (left) and the anonymized public data (right). The anonymized data follows k-anonymity with $k = 2$ in respect with Gender, $k = 2$ or $k = 3$ depending the age range. Therefore, even in case of disclosure of patients' name, age and gender it is not possible to link unequivocally each patient with their conditions. For instance, suppose that attacker knows both name, age, gender and public anonymized data. Trying to match a given condition back to the patient will result in more than one possibility for each condition. For instance, for the first anonymized row, both

Anna, Emily or Adam are candidates to match the condition Hypertension, thus $k = 3$. Other example, are female individuals aging from 45 to 65 years old, suffering from Diabetes. There are two possible matches: Karen and Catherine.

3.3.2 K-anonymity in Location Services

Due to its simplicity, K-anonymity has also been used for protecting users' locations in privacy-preserving Location Based Services (LBS) [48]. The idea is to achieve k-anonymization of location queries by preventing the LBS from identifying a single sender on some specific location based on received queries. Users need to send their location query, which contains a user identifier, its position and the time, to the LBS. To accomplish user location privacy, a Central Anonymity Server (CAS) removes the user identifiers, and obfuscates the location-time pair in their queries in a window (called cloaking region). This window is large enough for containing several users.

3.3.3 K-anonymity in Social Networks

More recently, due to the immense popularity of social networks, companies were demanded to protect users' personal information available on these platforms. Protecting user's privacy is not just a question of removing personal identifiers before publishing. Other information can be used to address user identification. In fact, it is possible to identify individuals based on their neighborhood (e.g. based on social friends list). Campan et al. [49] proposed an anonymization approach for social networking data. The idea is to use the K-anonymity model not just to protect the users' attributes (personal information) but also users' structural information (neighborhood information). Without neighborhood information an attacker cannot rely on neighbours (friends list) to infer the identity of a target user.

3.3.4 K-anonymity in Big Data

K-anonymity is also used in big data computation [50]. Today we generate more and more information that contains personal and private details about us. These data must be safely secure, preventing confidentiality leaks, integrity violations and it should remain available. One major points of big data is to perform useful queries and therefore use that data to enhance user experience on a given application or service. One possible approach could be to fully encrypt all data related to a every user of a given service in order to ensure privacy protection. However would be impractical to query that information later on. K-anonymity is then a key element, useful to cluster personal information in groups in order to query these data in an efficient way and at the same time preserve privacy.

3.4 Discussion

As presented in Sections 3.1 and 3.2, Tor is vulnerable to traffic correlation attacks whereby an adversary with the ability to intercept entry and exit nodes is able to identify the initiator of the communication. To reduce the chance of attacks, one approach is to avoid unsafe relay nodes, e.g., by deploying trusted relays on the client endpoints, continuously scan and advertise bad relay nodes, or enforce reliable guards by limiting the rotation period.

However, although avoiding unsafe relays can decrease the probability of flow correlation attacks, such defenses can hardly cope with a large AS-level adversary. Such an adversary is able perform flow correlation by intercepting the Tor traffic, and thus precluding the need to hijack or deploy relay nodes. To deal with AS-level attackers, the literature suggests some ways for clients to avoid potentially unsafe ASes, e.g., by monitoring routing information. Nevertheless, it is not trivial nor efficient for clients to detect BGP changes.

Alternative strategies to reduce the chance of falling prey to correlation attacks is to employ geographical avoidance, which translates in avoiding paths belonging to certain regions. Ultimately, however, these defenses cannot withstand a state-level adversary that can observe the traffic of the entire Tor network. This weakness results from a fundamental design decision whereby Tor offers unicast point-to-point channels: one single sender is connected to one single receiver throughout a circuit.

Our goal is then to enhance Tor so as to provide some degree of anonymity in the extreme situation of facing a state-level adversary. Our idea is to borrow the notion of K-anonymity proposed in multiple other domains for privacy-preserving applications. To the best of our knowledge, we are first to incorporate this anonymity definition into Tor so that a single circuit can be connected to multiple K senders, each of them could be equally responsible for initiating a circuit. We plan to achieve this by developing an add-on system to Tor that does not require the modification of existing Tor protocols or software components.

Summary

This chapter has detailed the proposed solutions in the literature to reduce or mitigate traffic correlation attacks by discussing their strengths and limitations. Their main approaches rely on preventing users from selecting unsafe relays or autonomous systems entirely. The chapter further introduces the notion of k-anonymity and briefly discussed the current application areas, ranging from databases to social networks and big data. Lastly, we discuss the main drawback of each solution and present an alternative method to dodge flow correlation attacks. In the following chapters we are going to present the design of TorK, our proposed system for building K-anonymous circuits for the Tor anonymity network.

Chapter 4

Design

In this chapter, we present the architecture of TorK, our proposed system for enhancing the security of Tor against a state-level adversary. This is achieved by supporting K-anonymous communications through the introduction of a new communication primitive named K-circuits. Next, we start by providing an overview of our system. Then we describe TorK's defense mechanisms to cope with several potential attacks of growing complexity, namely: protocol level attacks (Section 4.2), traffic pattern attacks (Section 4.3), bridge impersonation attacks (Section 4.4), and client-side Sybil attacks (Section 4.5).

4.1 System Overview

This section presents an overview of TorK. First, we describe the architecture of our system illustrating its components and usage scenarios. Then we present the semantics of TorK's novel communication primitive (K-circuits) in more detail, and clarify the threats that our system is designed to withstand.

4.1.1 Architecture

TorK's main goal is to protect Tor users' anonymity from traffic correlation attacks by offering K-anonymity protection. Specifically, TorK can prevent a state-level adversary from unambiguously identifying the IP address of a Tor client from amongst a group of K equally plausible candidates. This is achieved by enabling K cooperating users to voluntarily form a so called *k-circuit* by which all the K participants connect to an entry point in the Tor network but only one of them establishes a real circuit and leverages that circuit to truly transmit and receive data to some remote destination.

Figure 4.1 conveys this idea by depicting a deployment of TorK in a simple usage scenario involving three users. The system itself consists of two components which interact with the Tor network: a *gateway* and a *bridge*. The gateway is a software that runs on the users' local computers and acts as a client toward the bridge. The bridge, in turn, consists of a standalone server that accepts connections from one or multiple client gateway instances. Gateway and bridge implement a Tor pluggable transport and work together to exchange traffic between the Tor client software running on the users' computers and

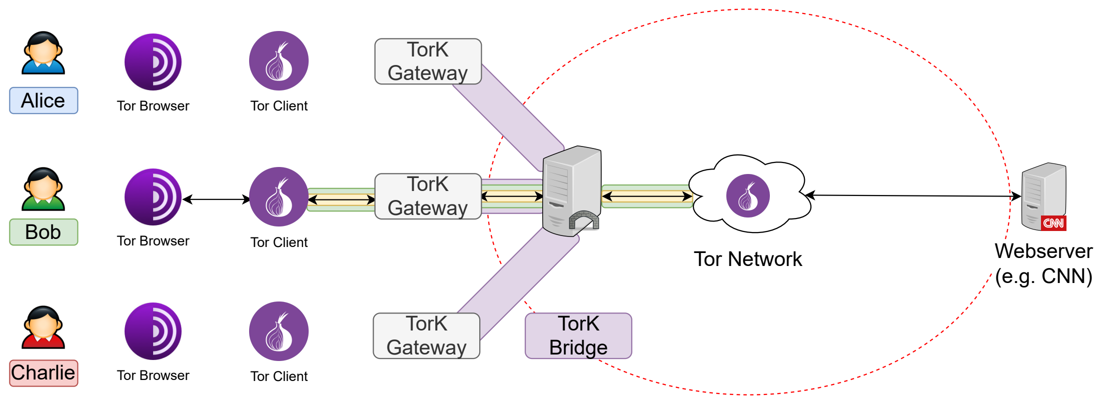


Figure 4.1: TorK architecture representing a k-circuit. All hops are observed by a state-level adversary – its domain of influence is indicated by the dashed red line. Tor onion layers are colored.

the Tor network. By leveraging the pre-existing Tor infrastructure and APIs – i.e., bridge and pluggable transports – TorK is fully compatible with the Tor protocols and software.

To explain how TorK works consider first how Bob leverages our system to access a news site through a standard Tor circuit (see Figure 4.1). As usual, this circuit is made up by three relays – entry, middle, and exit – and the client the middle node via a TorK bridge, which acts as an entry node. Assuming the existence of a state-level adversary that can observe all network packets exchanged in this communication, using simply a standard vanilla Tor circuit, the adversary would be able to deanonymize Bob's client by launching a flow correlation attack. TorK prevents this attack by allowing $k-1$ additional users – in the figure, just Alice and Charlie – to collaborate with Bob so that the adversary will not be able to distinguish who amongst them is the real originator of traffic associated with this circuit.

To achieve this, prior to the circuit establishment phase, all the k users initiate a communication channel between each local client and the bridge. We call each of these channels a *segment*. Each segment creates a tunnel between the local client and the bridge such that the local client can send arbitrary traffic through it. However, while Bob's segment will be used to transmit real data – i.e., the packets of the Tor circuit – the segments of the supporting users will transmit dummy payload, i.e., chaff. The bridge provides that the chaff is discarded and only Bob's actual traffic is forwarded to the middle node, thus entering the Tor Network where it is propagated to its destination.

To prevent an adversary from distinguishing which segment carries the Tor circuit's cells (by observing differences in the volume and timing properties of the traffic), the traffic of all participating segments is encrypted and modulated according to a common traffic shaping function. Thus, even though the attacker is able to capture and inspect the traffic from each of the three users (red ellipse) it cannot correlate the message to the original sender, i.e. from the three users discover the one that is transmitting the messages due to *indistinguishability* between segments. The attacker can inspect all hops in the network but he would have to randomly guess who the sender is, having a $1/k$ probability of succeeding. In virtue of these properties, TorK offers Tor users a new defense mechanism that provides K-anonymity. Next we introduce a new and simple communication primitive for Tor that abstracts out this semantics.

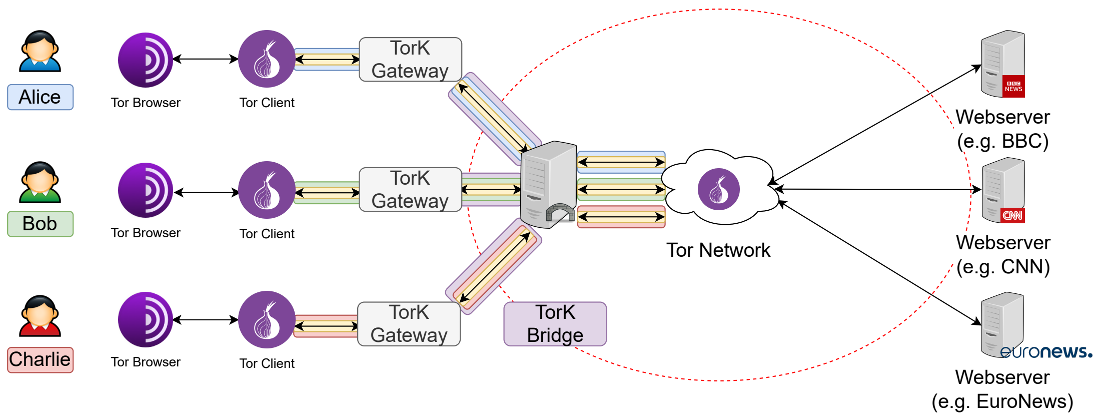


Figure 4.2: TorK architecture representing a k-circuit. All hops are observed by the attacker – dashed red line. Tor onion layers are colored. All three members opened a Tor circuit.

4.1.2 K-circuits Specified

TorK provides K-anonymity for Tor by allocating groups of size K. We use the term *k-circuit* to refer to a coordinated setup of K segments for tunneling Tor circuits owned by a group of K users. The specific k value to be used for a given k-circuit will be determined by the TorK bridge whenever a set of restrictions imposed by the group of potential participants is met.

Specifically, each client interested in participating in a k-circuit will be able to specify a lower bound k-*min* to the minimum number of participants that must be present in the k-circuit. This value allows each user to indicate the smallest acceptable anonymity set for his communications. For instance, Alice may indicate that she is only interested in joining k-circuits with no less than k-*min*$= 3$ users, i.e., $k \geq k$-*min*. The TorK bridge will then maintain a pool of participants in a waiting line until their respective k-*min* value restrictions are met. As soon as a group of k of participants is available such that k is smaller or equal than the k-*min* value restriction of each participant, then a new k-circuit can be formed for that group.

Although the k-circuit in Figure 4.1 provides a tunnel for a single Tor circuit owned by Bob, TorK allows for multiple circuits to be tunneled through. Specifically, in a k-circuit, TorK is able to forward more than one Tor circuits. In fact, each TorK allows to forward up to k circuits, one constructed by each member. For instance, Figure 4.1 depicts a k-circuit, with $k = 3$, where only Bob constructed a Tor circuit. However, it is possible that Alice and Charlie also constructed a Tor circuit as shown in Figure 4.2. This is particular important ensuring that supporting clients (Alice and Charlie) can also connect to the Tor Network. Thus, without any additional resources used, it is possible to forward three Tor circuits using the same, already existent, k-circuit structure. This design optimizes network resources and creates incentives for mutual cooperation among users.

TorK also supports *public* and *private* k-circuits. Public k-circuits are created automatically by bridges. By default, users are placed in public k-circuits with other users. In contrast, private k-circuits are created and managed exclusively by users. To create a private k-circuit, a given user – i.e., the k-circuit's owner – must send an additional specific instruction in the first message it sends to the bridge. The bridge acknowledges by returning the ID of the private k-circuit. The owner will then hold the right to grant access permissions to other users. To exercise its right and invite other users to join the private k-circuit, the owner only needs to share the ID with the invitees. Unlike with public k-circuits, bridges do not add automatically any other clients to private k-circuits. Thus, if no other client, apart from the owner, joins the private k-circuit, the owner may wait indefinitely or until a pre-defined timeout expires.

4.1.3 Incentive Model

In order to form a k-circuit, it is necessary the simultaneous participation of several (k) clients. This brings the question about the incentive model that will drive the end users to participate. The most straightforward case is the one where multiple users wish to access the Tor network while benefiting from k-anonymity offered by TorK. In this scenario, they all have the incentive to participate in the k-circuit. As mentioned in the section above, once a k-circuit is formed, every participant is able to create its own Tor circuit and tunnel it through its specific segment. The TorK bridge coordinates the group of participants, and ensures that the Tor circuit traffic exiting the k segments is properly routed to the respective middle node.

A more involved case is when the number of users willing to use the TorK system on a given bridge is not sufficient, i.e., it is less than k. In this case, we propose four strategies to satisfy this condition: *consolidation*, *invitation*, *remuneration*, and *donation*. Consolidation consists in redirecting connection requests arriving at different TorK bridges to a smaller number of bridges such that the number of connections at the same bridge reaches k. Invitation can be used if the number of existing pending connections is not sufficient to satisfy k. In this case users may ask trusted third parties (e.g., their friends) for help by establishing dummy segments (which will not carry any data). Remuneration covers the case where third parties can be hired in public forums and require a monetary compensation in exchange for their contribution – e.g., recruited through crowd sourcing services such as Amazon Mechanical Turk. Lastly, donation follows the same vein as projects like SETI@home and consists in the free contribution of resources by public volunteers. Volunteers are requested to install TorK client software on their computers, running in background or in idle times, which will connect TorK bridges and act as dummy k-circuit participants.

4.1.4 Threat Model

We have designed TorK to be secure against a state-level adversary whose main goal is to break the K-anonymity property in any given k-circuit. This property will be broken if the adversary is able to infer or guess with a probability higher than 1/k that some given participant in an established k-circuit is the true sender and / or receiver of traffic transmitted by an egress Tor circuit of the k-circuit in question.

To attain this goal, we consider an adversary with the following capabilities. The adversary has complete access to the global network infrastructure of the Tor network. Specifically, it has access to all the network links and middleboxes that interconnect the hosts of the Tor ecosystem: Tor users' computers, TorK bridges, Tor relays, and any remote host contacted by Tor users. Having access to this network infrastructure, the adversary is able to eavesdrop on the network and intercept any massage that has been exchanged between the communicating parties. This capability allows for launching passive attacks, e.g., through DPI or statistical traffic analysis. Additionally, the attacker is also capable of dropping, delaying, modifying, duplicating, or injecting packages into the network. These skills allow for mounting active attacks, e.g., disrupting the communication of one of the segments of a k-circuit.

Based on the aforementioned capabilities, we consider the following main attacks to be in scope:

Protocol level attacks: The adversary may try to deanonymize the author of the communication within a k-circuit by exploiting vulnerabilities in the communication protocol between the TorK gateway and the TorK bridge, namely, (i) by reading unencrypted fields of exchanged packets, or (ii) by detecting or inducing flaws in the synchronization between gateways and bridge. E.g., if a participant can create a Tor circuit before all other k-1 participants have joined, it will be evident who the real data transmitter is.

Traffic pattern attacks: The adversary may attempt to correlate the traffic being transmitted through the egress Tor circuit of a k-circuit with the traffic generated by its individual segments. Several distinct features can be used for this purpose, e.g., volume of traffic, inter-packet arrival time, or packet lengths.

Bridge impersonation attacks: The adversary may deploy TorK bridges that will be fully under its control and advertise its services to potential victim users. As in a typical honeypot setup, users that connect to that bridge will be vulnerable to deanonymization since the adversary has full access to the memory address space of the TorK bridge process. As a result, it is possible to monitor in real time all the messages sent by each participant. From the content of these messages it is possible to identify the real sources of data, e.g., by checking which packets carry chaff or useful data.

Client-side Sybil attacks: The adversary may also deploy Sybil clients and having them form k-circuits with legitimate TorK users. In the worst case, if the adversary is able to control $k-1$ participants of a k-circuit, then the remaining participant p will be trivially identified. All the adversary needs to do is sending chaff through all its controlled participants and let p initiate the Tor circuit.

4.2 Thwarting Protocol Level Attacks

To assure that K-anonymity is preserved, TorK needs to coordinate the action of several users joining a k-circuit. Without proper coordination, it would be trivial for an attacker to exploit different users' behaviors. For instance, observing the Tor network allows an adversary to determine who among the k users are actually sending traffic, even though the traffic is encrypted. To this end, TorK constructs groups of k users designated as k-circuits and uses a custom protocol to coordinate the actions of its members. Next, we present a protocol for the communication between clients (i.e., TorK gateways) and TorK bridges which is robust against protocol level attacks. After introducing this protocol, we describe the respective state machines implemented by each endpoint.

4.2.1 The TorK Protocol

In this section is detailed the protocol between two TorK gateways and bridges. This protocol aims at coordinate clients in order to guarantee k-anonymity among all clients connected to a bridge.

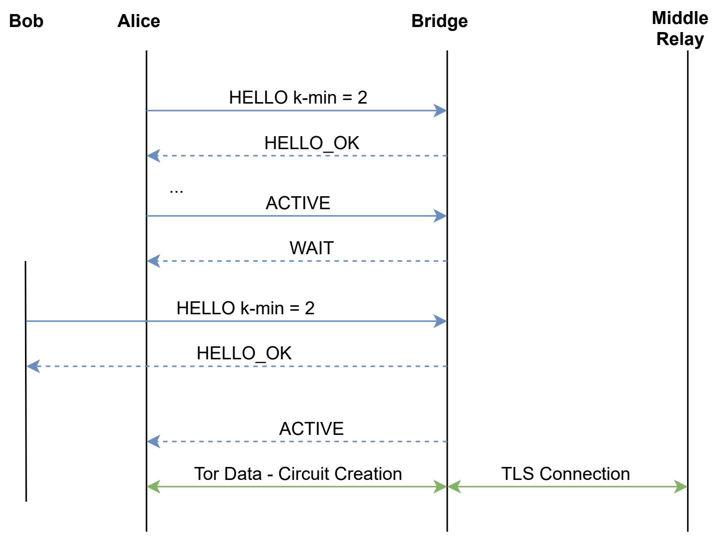


Figure 4.3: TorK gateways and bridge message exchange.

K-circuit establishment request: When a client connects to a bridge, the TorK protocol begin. Figure 4.3 presents the messages exchanged between two clients and a bridge. Initially, each client gateway, upon connection to the bridge, sends a HELLO message containing the specification of its k-min restriction value. In other words, gateways define the minimum number of users that should be connected at the bridge in order to establish a circuit. The bridge validates this number and reply with HELLO_OK or HELLO_ERR in case of success or invalid k-min value, respectively. In the case of Figure 4.3, Alice requests a minimum of two users that must be present within the same k-circuit.

Tor circuit creation request: Later, when a client intends to open a Tor circuit it will ask the bridge by sending an ACTIVE control frame. Bridges will reply either with ACTIVE when all k-circuit members' k-restrictions are hold, and consequently, the client is authorized to open the Tor circuit, or with WAIT otherwise. Receiving WAIT instructs the client to wait for a further response. Eventually an ACTIVE will be sent by the bridge when all restrictions hold. Since Alice requested two clients and only one, herself, is connected at this moment, she will wait until another client is placed in the same k-circuit.

K-circuit ready: Bob joins requesting also a k-min of 2. At this instant, bridge has two connected clients and therefore Alice has authorization to create a Tor circuit since their restriction is now fulfilled. To concede authorization, the bridge sends an ACTIVE to Alice. From this point onward, Bob is authorized to send data frames towards the bridge, containing Tor circuit cells, and consequently, he is also allowed to open a Tor circuit, which is going to be forwarded to the Tor network.

Reaction to changes in k-circuit conditions: At any time, an active client can receive a WAIT frame. Such indication tells clients that their restrictions, despite having been satisfied in the past, they do not hold currently anymore (e.g., due to a participant that has left). Clients react by destructing the circuit immediately and wait further to create a new circuit. Apart from WAIT frame, clients can receive a CHANGE frame. Such frame requires that client replace their current circuit. In other words, teardown the existing circuit and open another one. Upon sending a WAIT or CHANGE control frame, bridges immediately close clients circuit towards the Tor network and drop any data frames received

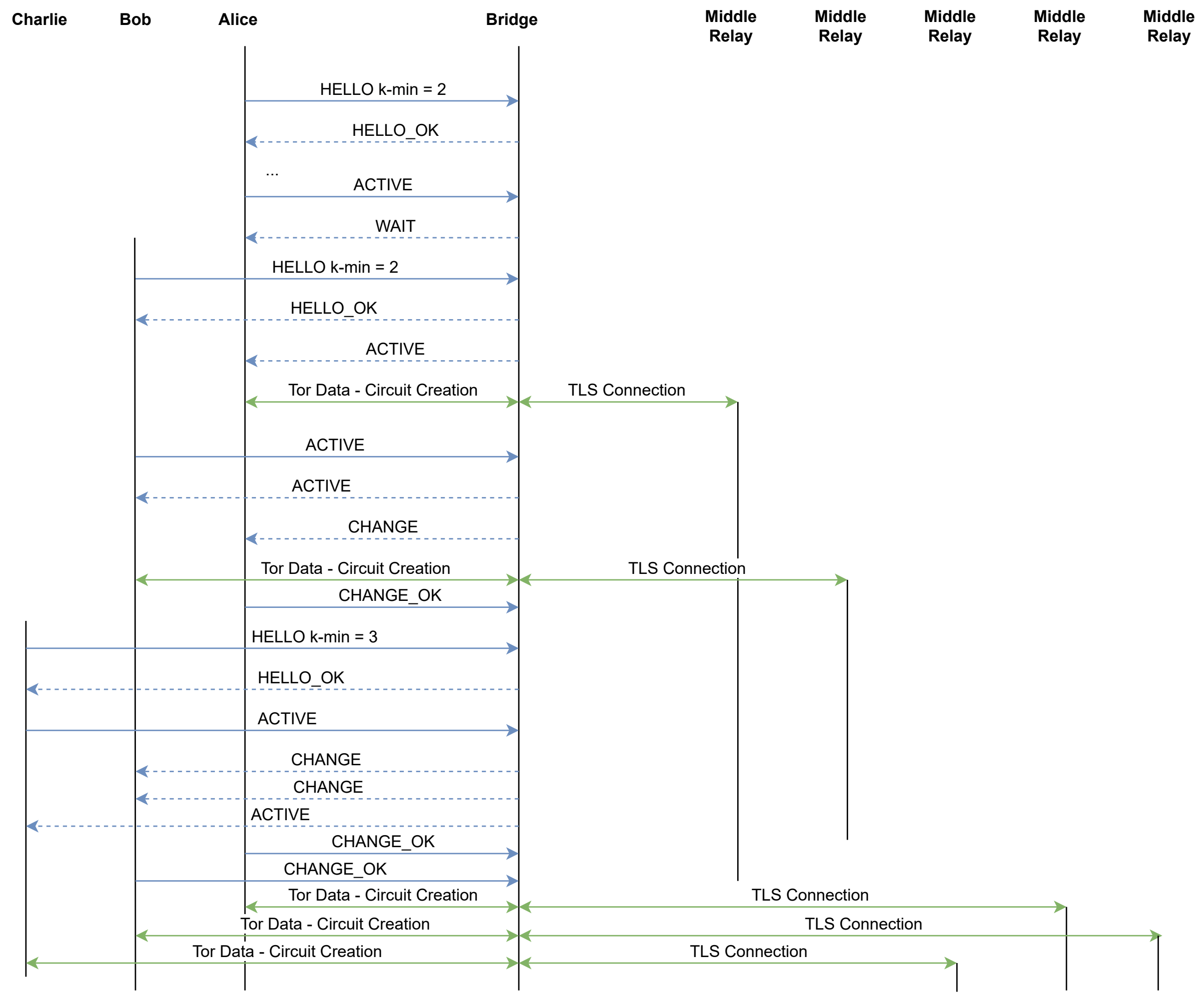


Figure 4.4: Detailed example of TorK protocol and their exchange messages.

after sending the control frame. In case of changing circuits, clients should acknowledge bridges with a CHANGE_OK in order to restore the dropping of data frames. The strategy is to prevent delivering Tor cells from previous Tor circuit (towards the Tor network) after the client restriction became broken.

Tearing-down circuits: However, client gateways can tear-down the circuit by their will or give up of waiting for bridge authorization. Active or waiting clients can send an INACTIVE to indicate that they will tear-down their circuit or they no longer intended to open one, respectively. For example, Figure 4.4 depicts a scenario with initially, two clients (Alice and Bob) requesting k-min of two. Alice also requests to open a Tor circuit having to wait for at least another client. Alice and Bob change/construct a circuit when signaled by the bridge. Later a third client, Charlie, connects to the bridge requesting a k-min of three and intend to create a circuit. Since both Alice and Bob had already an open circuit, continuing using those circuits will deanonymize Charlie. This occurs since an attacker can perceive that one of those two circuits belong to one of those two clients, initially even though it cannot pinpoint exactly to whom each circuit belong. If a third client constructs a new circuit and the remaining continue to use the current circuit, then the attacker can conclude that the new circuit's author is the third client, Charlie. Thus, to prevent such flaw, Alice and Bob receive an order to change their circuits. Therefore an attacker would see the destruction of the current two circuits and the creation of three new ones without knowing

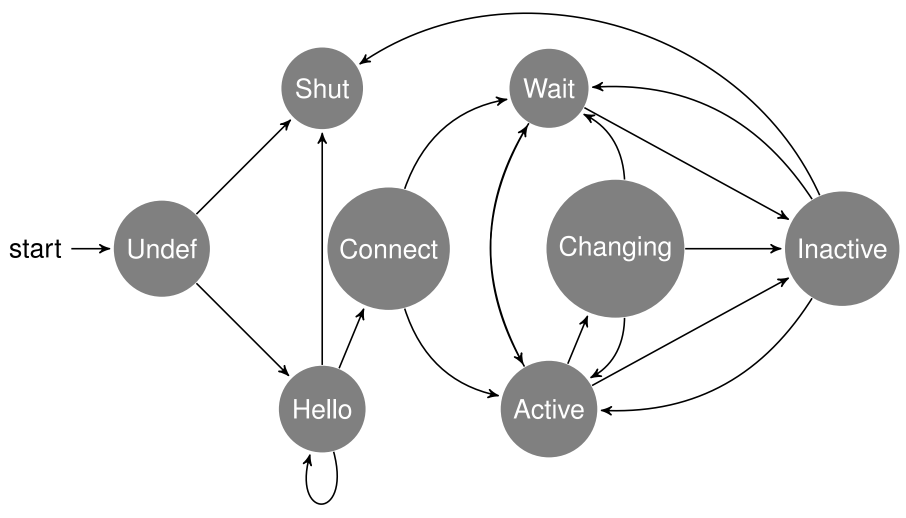


Figure 4.5: TorK Protocol: Client State Machine

exactly to whom each new circuit belong.

Advanced scenario: Additionally, TorK should handle efficiently the case where one client leave one k-circuit, and one or several other members restrictions become broken. Designing an algorithm for such scenario is not trivial and can be classified as a constrained optimization problem. We can formulate the problem as follows: given a set of users in a pool P with a k-min restriction, how to create and organize K groups maximizing the number of users whose restriction holds without breaking any k-min restriction.

There are two main approaches to this problem: either the broken members wait until one or several new members enroll in that incomplete k-circuit, or those broken members are redistributed among other k-circuits. Waiting for new members to enroll can be beneficial when a significant fraction of the users are constantly joining and leaving the network. In other words, waiting for a reasonable amount of time is opportune since, eventually, every k-circuit will be complete. Yet, if the number of users are stable and they establish long term connections, then it is desirable to redistribute the pending users among other k-circuits. Otherwise, pending users will wait for long periods of time. Ideally, the best approach would be a hybrid solution, where bridges gather users' statistical information from the past minute, five minutes, etc... and choose, among the two approaches, the one that fits accordingly. Currently we adopt the first approach, although we envision that a hybrid solution should be explored in future work.

4.2.2 K-Circuit State Machines

Internally, bridges and client gateways maintain a state machine to coordinate all peers belonging to the same k-circuits. Bridges keep track of the state of each client and they are responsible to ensure traffic policing, this is, that clients should only send traffic when all k-min restrictions are met. Clients only transmit real data when the bridge ensure that all k-min values, from all connected peers, are fulfilled.

The client state machine is depicted in Figure 4.5. Client gateways start in a undefined state (UNDEF) and start by connecting to the bridge and beginning the HELLO phase. The CONNECT state matches

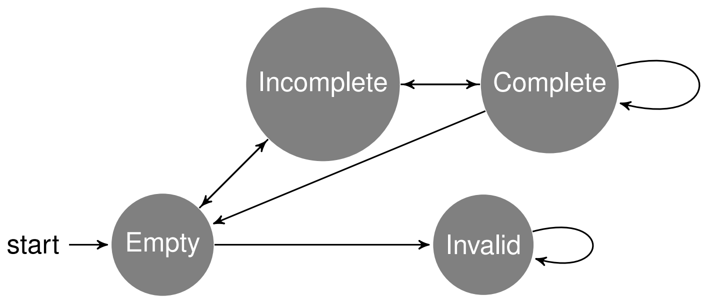


Figure 4.6: TorK Protocol: Bridge K-Circuit State Machine denoting K-Circuit status (σ).

the moment after the HELLO phase occur and after client requests to open a circuit, i.e. sends a ACTIVE control frame. The CHANGING state describes a state where a client received a CHANGE control frame and is tear downing the current circuit and opening a new one. The remaining states are in line with the protocol described in Section 4.2.1.

Figure 4.6 depicts the state machine of k-circuit state managed by the bridge. The state, in a given instance, will depend solely on the number of members as well as the restrictions defined by the members. For instance, if Alice and Bob define both k-min of two, and a *Empty* k-circuit is created by the bridge. Their k-circuit will be *Incomplete* when only one member is present. When both users are present, the k-circuit become *Complete*. Exceptionally, in case of unrecoverable situations (e.g. both define invalid k-min), the k-circuit can become *Invalid*.

Bridges need to implement a K-Circuit policy for creation, deletion and, optionally, reordering users in k-circuits. Such algorithm will chose where to place a new connected user among a set of already existing k-circuits. The pseudocode is presented in Algorithm 1. The strategy used to select where to place a new connected client p is described by the KCIRCUIT_CREATION. The function starts by placing the new client p in k-circuits whose state σ is incomplete, if any. An incomplete k-circuit K_i contain one or more clients whose k-restriction do not hold due to number of members ($\#K_i$). Therefore, the function checks if all k-circuit are complete and have the maximum number of members (MAX_USERS). If there are more than one incomplete k-circuit, the function chooses the closest k-circuit to became complete. In other words, if one incomplete k-circuit needs three users and a second incomplete k-circuit needs

Algorithm 1 Public K-Circuit Creation strategy pseudocode.

```
1:  function KCIRCUIT_CREATION($P, K, p$)
2:      if $\#K = 0 \vee \forall K_i \in K : \sigma(K_i) = Complete$ then        ▷ All k-circuits are complete
3:          $K_m = \min_{K_i \in K}\{\#K_i\} \wedge \#K_i <= MAX\_USERS$
4:          if $K_m = \emptyset$ then
5:              $K_c =$ CREATECIRCUIT
6:              $K_c \leftarrow K_c \cup p$
7:              return $K_c$
8:          else
9:              $K_m \leftarrow K_m \cup p$
10:             return $K_m$
11:     else                                                    ▷ There are incomplete k-circuits
12:         $K_m = \min_{K_j \in K}\{\#K_j - \max\{K_{min}(K_j)\}\}$
13:         $K_m \leftarrow K_m \cup p$
```

one user, the function will place the new connected client in the second k-circuit. Yet, if all k-circuits are complete and there is at least one contain less than MAX_USERS then the new client is placed in that k-circuit. The rationale of this strategy is to have leftover clients, thus if one or more decided to leave, the remaining members may still form a complete k-circuit. Otherwise, if all k-circuits are complete and have MAX_USERS then the new connected client is placed in a brand-new k-circuit.

4.3 Thwarting Traffic Pattern Attacks

In addition to be secure against protocol level attacks, TorK must also be able to resist against traffic pattern attacks. In this section, we start by explain the two main facets of these attacks and then present our solution which involves dynamically throttling the bandwidth of k-circuits' segments.

4.3.1 Two Faces of the Same Problem

Resistance to passive attacks: To achieve K-anonymity, the communication channels that interconnect the participants of a k-circuit and the bridge – i.e., the segments – must share the same properties irrespective of the payload that is tunneled through them, i.e., chaff or Tor circuit cells. In other words, the segments must be *indistinguishable*. In particular, we must ensure that the characteristics of the payload tunneled through each segment do not influence the external characteristics of the each segment's network packets, otherwise information about the payload could be leaked to a network eavesdropper. However, providing this guarantee can prove itself to be challenging. Some recent advances in traffic analysis [51] show that it is possible to detect with high degree of accuracy subtle differences between flows and leverage these differences to defeat state-of-the-art censorship-resistant communication tools. TorK must be robust against such techniques, which essentially leverage machine learning in order to identify features that can be used to differentiate both types of segments.

Resistance to active attacks: It is also necessary to shield TorK against potential network-level active attacks. For instance, consider the example depicted in Figure 4.1. Assume also that the bridge is trusted and all cooperating clients – say, Alice and Charlie – are benign users. In this circumstances, notwithstanding, an adversary with the ability to drop, inject, or modify the sequence of network packets could try to deanonymize Bob's communication by instrumentally dropping packets of each segment of the k-circuit, and observe for which of the segments there is a visible reduction in the throughput of the bridge's outgoing connection to the middle node. In this particular case, dropping packets in Bob's segments will interrupt the transmission of Tor circuit cells which will be reflected in a smaller debit of packets exiting the bridge and allow the adversary to identify the real sender: Bob.

4.3.2 Encryption, Shaping, and Bandwidth Control

To thwart the aforementioned attacks, we employ a combination of techniques. To deal with passive attacks, all packets transmitted between each client and the bridge are encrypted and reflect the packet

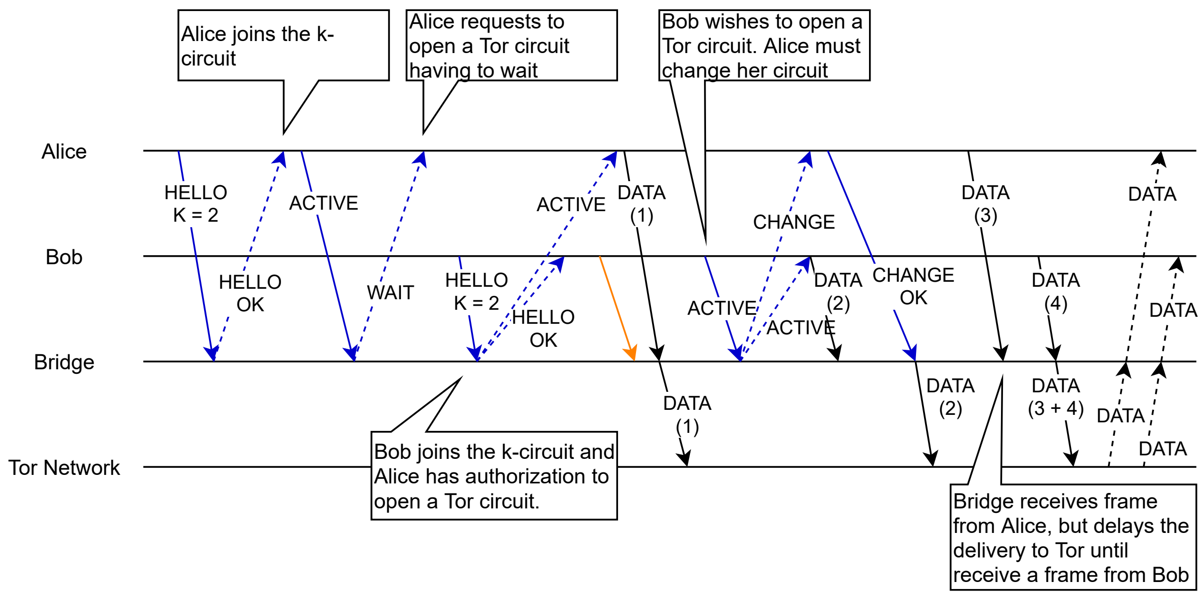

Alice joins the k-circuit
Alice requests to open a Tor circuit having to wait
Bob wishes to open a Tor circuit. Alice must change her circuit
Alice
Bob
Bridge
Tor Network
HELLO K = 2
HELLO OK
ACTIVE
WAIT
HELLO K = 2
HELLO OK
ACTIVE
DATA (1)
DATA (1)
ACTIVE
CHANGE
ACTIVE
DATA (2)
CHANGE OK
DATA (2)
DATA (3)
DATA (4)
DATA (3 + 4)
DATA
DATA
DATA
DATA
Bob joins the k-circuit and Alice has authorization to open a Tor circuit.
Bridge receives frame from Alice, but delays the delivery to Tor until receive a frame from Bob

Figure 4.7: Bandwidth control measure in a k-circuit with k = 2. (Only relevant chaff messages are presented - orange).

size and inter-packet time distributions of a common data stream (e.g., a video) which will serve as a cover for tunneling chaff / data between clients and bridge. The choice of which cover signal to use for modulating the traffic needs to be studied carefully because of the involved trade-off between network efficiency – i.e., the amount of wasted bandwidth when no useful data is sent – and performance – i.e., the amount of available bandwidth within each segment for sending useful data. To strike a balance between these two conflicting forces, it is necessary to study different traffic shaping techniques that can yield good results in our problem domain, e.g., traffic morphing [7] or a technique proposed by Piotrowska et al. [52] which uses Poisson mixing to achieve both sender and receiver anonymity and unobservability. We performed an empirical study with different traffic shaping functions whose results are described in our evaluation.

To prevent an attacker from correlating TorK frames with the Tor circuit by observing bursts in the bandwidth of clients, bridges implement a bandwidth protection mechanism. Bridges only deliver Tor traffic to the Tor network after receiving a frame from all connected clients. Two Tor cells received by different clients within the same k-circuit having different bandwidths (e.g. 200Kbit/s and 1Mbit/s) are drained to the Tor network at the speed of the slowest client. In this way, an attacker cannot correlate a client to its Tor circuit based on timing properties even when delaying clients connections deliberately. In this way, this mechanism can also mitigate the active attacks described in the section above.

Consider the example depicted on Figure 4.7, that displays Alice and Bob, two clients in a k-circuit of two. For simplicity we only present relevant chaff frames, depicted as orange. Alice starts the TorK protocol defining her k-circuit and requesting permission to open the circuit by sending the ACTIVE control frame to the bridge. Later, Bob also starts the connection and defined the value of k. At this point Alice can open a circuit to send and receive data. The bridge's bandwidth protection ensure that DATA frames are only dispatched to the Tor network upon reception of at least one frame from every client, regardless its type. In this case, Alice sent DATA frame (1), and since bridge received a CHAFF

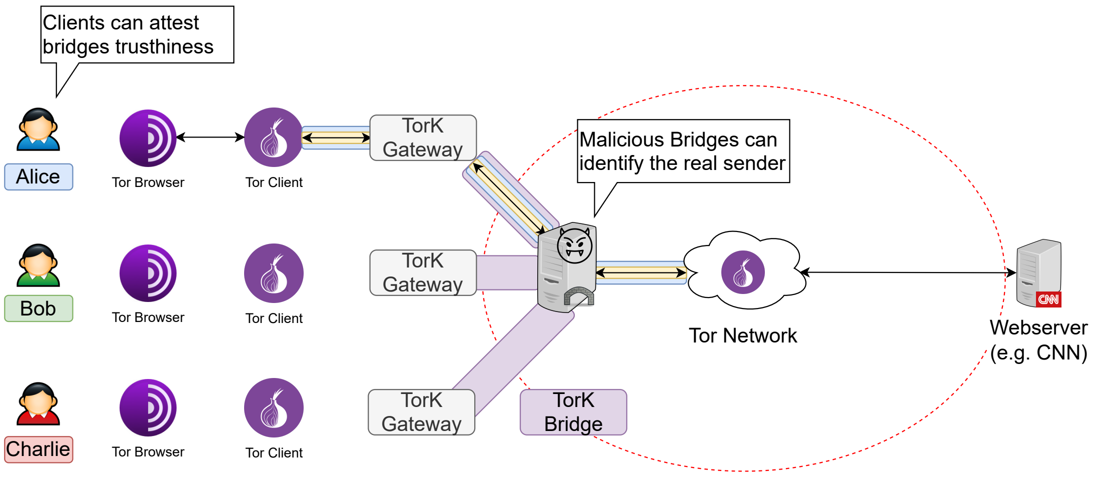


Figure 4.8: Scenario where the TorK bridge is controlled by an attacker.

frame from Bob it is possible to dispatch the Alice's data – DATA (1) – to the Tor network safely. Later, Bob requests to open a circuit, which causes Alice to change her circuit and Bob to open a circuit. Bob now sends Tor cells along a DATA frame – DATA (2). However DATA (2) dispatch to the Tor Network will be stalled until the reception, in this example, of a CTRL frame from Alice – CHANGE OK. As soon as the bridge have received a frame from both clients, the stalled data frame – DATA (2) – is sent to the Tor Network. The same behavior occurs when both Alice and Bob send data – DATA (3 and 4).

4.4 Thwarting Bridge Impersonation Attacks

So far we have covered mostly TorK's defenses against network level attacks. It is also reasonable to assume that a powerful attacker may deploy malicious bridges aimed at deanonymizing clients that opt to use it. Given that the Tor network is open, anyone can deploy TorK bridges, including malicious ones that can be leveraged to deanonymize end users. To mitigate these threats, we propose to leverage trusted hardware technology.

The anonymity properties of k-circuits depend upon the fact that TorK bridges are trusted (1) *not to reveal* the identity of the participating clients, and (2) to *follow the protocol* expected to be implemented between client and bridge. Simply put, the execution state of a TorK bridge must preserve two main properties, respectively, confidentiality and integrity. It follows then that it is necessary to ensure that both these properties are guaranteed by the time a client decides to connect to a given TorK bridge. This will allow the clients to assess whether the bridge is trustworthy or not.

To detect potentially malicious TorK bridges (as depicted in Figure 4.8), we propose a solution based on trusted computing. Inspired by the approach of Kim et al. [53], our solution requires that the bridge is equipped with trusted hardware with the capability to instantiate a trusted execution environment (TEE). A TEE provides an protection domain where the execution state of the guest program is guaranteed to run in isolation from the host's OS, and therefore cannot be tampered with by the system administrator. Moreover, a TEE provides mechanisms for measuring the integrity of the guest program at bootstrap (i.e., by computing a cryptographic hash of the program's binary) and allowing for remote clients to attest

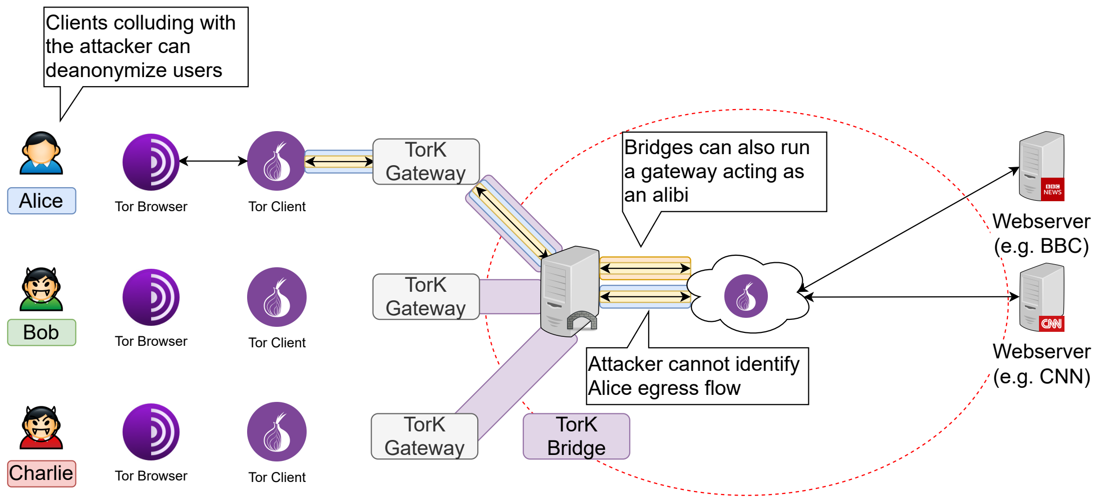


Figure 4.9: Scenario where the two TorK clients are controlled by an attacker colluding to deanonymize one user.

to the integrity of that program by checking a signature of the program's hash signed with a hardware key. The idea is then to secure the TorK bridge software inside the TEE and run a remote attestation protocol at the client side to verify that the bridge is trustworthy before the client can proceed with joining any k-circuits. If the program has been modified or replaced (e.g., to disclose the identify of clients), attestation will fail and the client will refuse to trust in that bridge.

To implement TEE, commodity Intel-based hardware offers two interesting supporting technologies. One possibility is to create the TEE using Intel's SGX technology. In this case, the TEE is instantiated inside special memory-protected containers called *enclaves*, and remote attestation is supported by a set of cryptographic keys deployed by Intel in all its SGX-compatible chips. Thus, we could run the TorK bridge software inside an enclave and verify the signature issued by the bridge against a valid SGX key certificate issued by Intel. However, a potential shortcoming of using SGX is that the overhead of entering and exiting enclaves is considerable, which could negatively affect the performance of the TorK bridge and hamper the latency of tunneled Tor traffic. An alternative approach would be to use Intel's TXT hardware extensions in combination with a secure cryptoprocessor named TPM. Using TXT, the TorK bridge software would be executed inside a hardware-secured virtual machine where it could run at native speed. The TPM would provide the hardware support for measuring the integrity of that VM, and issuing the attestation signature to remote clients. Both TXT and TPM technologies are prevalent in modern Intel-based platforms. In our work, we implement a solution based on Intel SGX and verify empirically that the performance overheads are relatively low (see Section 6.5).

4.5 Thwarting Client-side Sybil Attacks

So far we have assumed that all participants in a k-circuit are benign. However, if one or more participants are malicious and collude with an adversary that can monitor the network traffic, it may be possible to deanonimyze a legitimate client sending Tor traffic through that k-circuit. Consider for instance the example shown in Figure 4.9. If Bob and Charlie are malicious, they can inform the network eavesdropper that they are not sending any traffic through their specific segments. Given that the network eavesdropper can observe that the number of participants in that k-circuit is three and he can rule out Bob and Charlie, then the only possible source of the traffic exiting the bridge is Alice. Thus, in the general case, the security guarantees offered by TorK as it has been discussed up until this point is a function of the number of malicious clients m that can join any given k-circuit. Specifically, the probability of an adversary to guess the identity of a given sender is $1/(k-m)$, which means that the real sender can be deanonymized if the adversary is able to control $k-1$ participants of a given k-circuit.

To mitigate such attacks, we propose to investigate several different strategies, which can possibly be deployed and operate in combination. One idea is to implement more clever *participant selection policies* at the bridge. Essentially, in addition to satisfying the k-*min* value restrictions of participants, the bridge can be more selective of which participants to include in a given k-circuit so as to increase their geographical and / or AS diversity. This is based on the intuition that it may be more difficult for an adversary to control a large number of clients deployed across different ASes or geographic regions. A second idea is to implement *dynamic bridge assignment*, i.e., rather than letting the clients to unilaterally choose which bridge they connect to, participants can be automatically redirected to different bridges so as to increase participant dispersion and make it harder to launch targeted attacks on some specific bridge. Another idea is to provide support for access control policies based *user-level restrictions*, where users can specifically white-list or black-list certain participants that they deem to be trusted (e.g., belong in their circle of friends) or untrusted (e.g., are known to be operating from certain countries). User restrictions can also be implemented based on some reputation system for promoting well-behaved users. One last idea is to run an *alibi agent* at the bridge itself. Essentially, for each k-circuit, the bridge would run a piece of software that mimics the behavior of a real client by establishing Tor circuits and tunneling traffic through these circuits. As a result, even if the attacker could control all of $k-1$ participants, it could not know whether the true originator of the traffic exiting the bridge is the remaining participant or the alibi agent running on the bridge.

Summary

This chapter detailed TorK overall architecture, main entities and threat model. Next it was described how to managing k-circuits, from the recruiting process to its formation and synchronization. It is also detailed the TorK Synchronization Protocol coordinating K users in a circuit to ensure k-anonymity over indistinguishable segments and measures to withstand against network-level attacks. Next it was discussed some measures to thwart potential attacks that can be exploited by an active attacker, such as delaying packets deliberately. Lastly, it was discussed measures against malicious bridges and clients which can be deployed and launched by an attacker.

Chapter 5

Implementation

This chapter addresses the implementation details of our TorK prototype. Section 5.1 describes an overview of the implementation. Then, we provide several details about the implementation of TorK's main components, in particular the process of the indistinguishable channel, k-circuits and several measures to cope with active attacks describing also some practical decisions taken and challenges faced. Lastly, Section 5.8 describes the implementation of trusted computing measures to enhance TorK against hardware and software tampering.

5.1 Implementation Overview

One of our main goals was to implement TorK without changing the underling Tor infrastructure. Such goal decouples TorK from the Tor internal architecture improving code maintainability of both TorK and Tor. TorK must also be able to handling Tor circuits, mainly create, extend and destroy circuits in order to sustain the desirable properties against an adversary. Tor controller protocol [54] grants the control of Tor configurations, including the possibility to handling circuits and streams, to an external application. Overall, we have achieve this goal by implementing a fully functional prototype of TorK for GNU/Linux. TorK includes both a Client and Bridge (Server) component. The prototype was written in C++ using OpenSSL[1] and Boost[2] libraries.

Figure 5.1 describes a high-level overview of TorK. To initiate the service TorK must be running on both endpoints: in clients gateways and bridges. To achieve this, the Tor client running on the user's machine must be set up to use the TorK pluggable protocol. Upon starting, Tor launches a TorK instance who exposes a SOCKS [55] proxy to receive Tor traffic. Thus an ordinary application performs request to Tor, which in turn, passes them to TorK by the provided proxy, which is responsible to forward the Tor cells to the bridge. More specifically, when a connection is established by a client to the bridge, an indistinguishable channel is active. Such channel drains requests according to a specific traffic shaping function. When a client constructs a Tor circuit, the traffic (Tor cells) passes along TorK which in turn is converted into frames and then passed to the traffic shaping function. Bridges receive the shaped traffic

[1] `https://www.openssl.org/`
[2] `https://www.boost.org/`

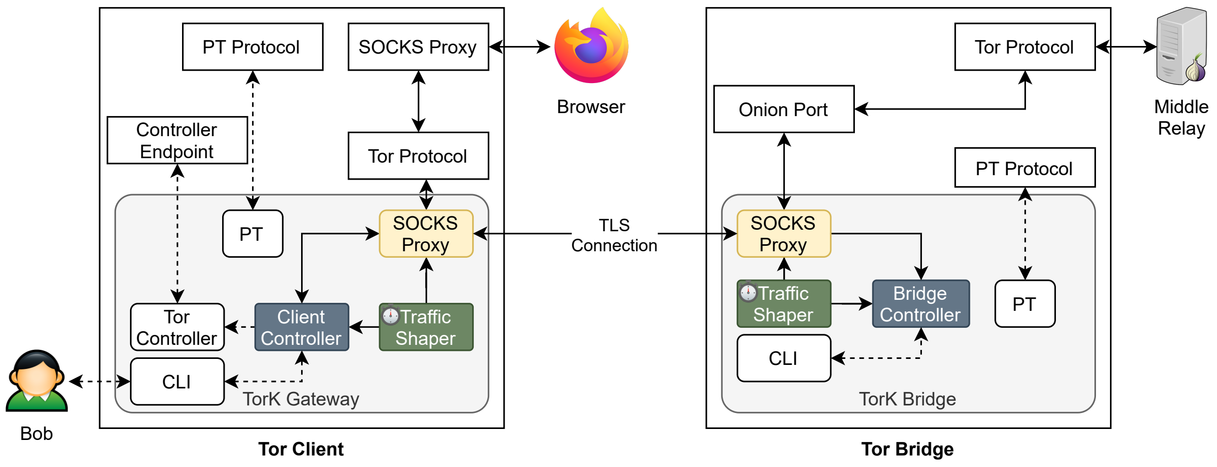


Figure 5.1: Overview of TorK's components with Tor.

and convert them to the original format before delivering it to the Tor network, i.e. convert TorK frames into original Tor cells. Apart from the traffic shaping function, TorK bridges employ a synchronization mechanism to coordinate users belonging to the same k-circuit.

TorK is structured in six different components. Each component whether aims at implementing an Tor API, in which TorK can use to setup Tor or implements TorK architectural components as described in Section 4: indistinguishable and unlinkable channels, k-anonymity through k-circuits, the TorK protocol, and finally, measures to withstand against active attacks. The following subsections present implementation details, issues and challenges faced for each component.

5.2 Pluggable Transport

In order to avoid changing Tor protocols we designed TorK as a Tor pluggable transports (PT) (introduced in Section 2.2). In particular, Tor authors published the PT specification. Such specification written in a RFC-like document, details how Tor starts and shutdown PTs and the environment variables that should be read and set. TorK implemented the PT API from scratch since there were only Go and Python implementations available online.

Each Tor instance loads a configuration file – denoted *torrc* – containing TorK executable path, command line arguments and Tor mode operation (e.g. bridge or client). When launched with this configuration file, Tor starts TorK as a subprocess. Several environment variables are shared by Tor, accessible to TorK, exposing Tor configurations, e.g. the bridge remote address. TorK reports back to Tor by writing success or error messages into the stdout in a standardized format, which Tor, the parent process, is able to capture and read, e.g. local port where TorK accepts connections from local Tor (sketched as the red/green arrows in Figure 5.2). During shutdown, Tor also takes care of a controlling shutdown TorK. This process is accomplished by closing the stdin of TorK. TorK, upon detecting a closed stdin, begin to shutdown cleanly all of its components. Apart from the initial configuration, is up to TorK to establish a remote connection with the other endpoint to send and receive Tor data.

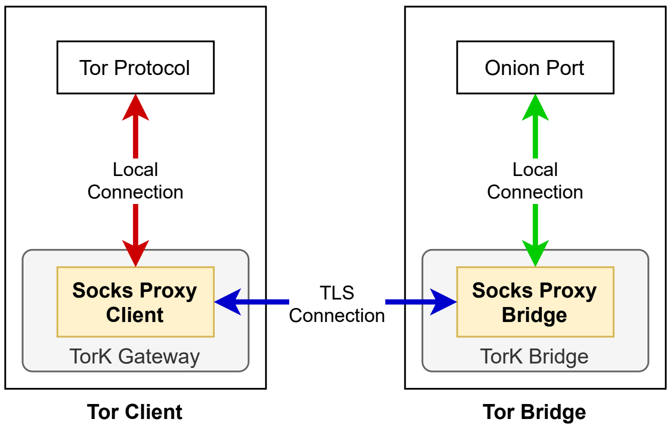


Figure 5.2: Pluggable Transport configuration and TorK Socks Proxy.

5.3 SOCKS Proxy

TorK gateway exposes a SOCKS [55] proxy as a endpoint for receiving Tor traffic. TorK bridges exposes a reverse SOCKS proxy that accepts local Tor connections. SOCKS protocol is a well known standard and optionally includes authentication. The circuit and session used by SOCKS make it a preferred interface for routing tools. Thus, Tor itself exposes a SOCKS port allowing external applications to run seamlessly on top of the Tor Network [30]. SOCKS instances are local to Tor, since it is running on the same host, thus Tor does not use SOCKS authentication.

Internally, the proxy (see Figure 5.2) was modified to maintain a pair of file descriptors corresponding to the local Tor connection and remote client/bridge connection (red/green arrows and blue arrows). Essentially, the proxy reads data at one end, passes the data to the Client/Bridge Controller for processing before sending the result to the other end.

Implementing some TorK protocol commands like CHANGE and WAIT required the current Tor circuits to be tear down was an arduous task. When bridges perceive that some client should change their circuit, they notify the client and block all outgoing data traffic from that client, preventing it from entering the Tor network and therefore, being visible to a network attacker. The blockade imposed by the bridge will drop every data frame and it will remain active until the client acknowledge by replying CHANGE_OK. By dropping Tor data cells we are inadvertently disrupting the counter values used in Tor cells TLS encryption. As a consequence, the connection is shattered upon restoring, i.e. upon lifting

```
void
entry_guard_failed(circuit_guard_state_t **guard_state_p)
{
  ...
  //entry_guards_note_guard_failure(guard->in_selection, guard);
  ...
}
```

Figure 5.3: Tor source code change to prevent marking bridges as down upon CHANGE operations. File: `tor/src/feature/client/entrynodes.c`

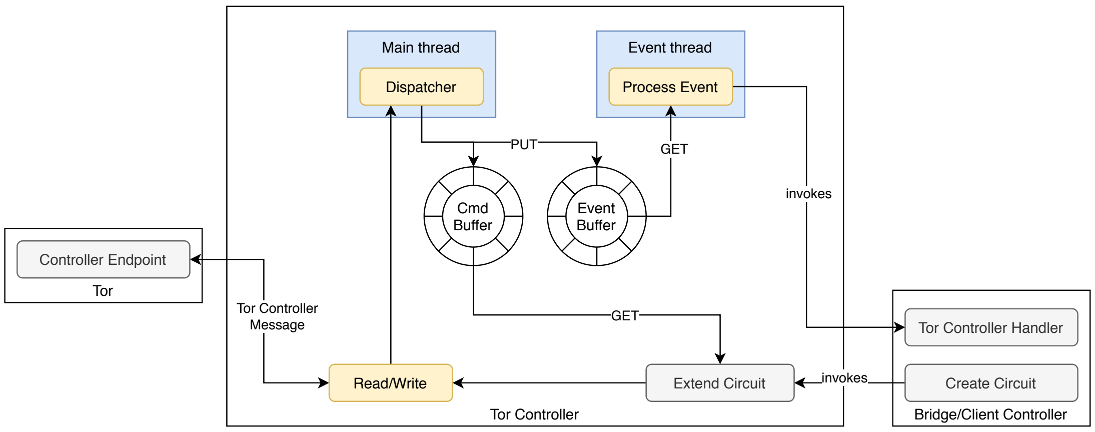


Figure 5.4: Tor controller implementation.

the blocking, mainly due to the reception of TLS Tor cells containing wrong/unexpected counter value. Recall that several previous TLS cells were dropped. Due to this drawback, we added the capability to shutdown only the local connections from each side and restore them in the future. In other words, changing a circuit required tuning the proxy to allow local connections in both ends to be reopened (red and green arrows), while maintaining the client to bridge connection (TLS connection, blue arrows).

The approach presented allows Tor TLS connections to be reopened, although another hindrance arises: Tor marks bridges as faulty when the local connection closes abruptly. Tor will not use the faulty bridge in the future. To this end, we needed to change one single line at Tor source code (Figure 5.3) to disable the process of marking a bridge faulty when connection abruptly closes, as the result of changing a circuit. Such modification do not change any Tor or OR protocol and, in the future, can be enabled/disabled by a Tor specific command line argument.

5.4 Tor Controller

Apart from receiving and modifying Tor traffic on-the-fly, TorK must be able to control the entire Tor circuit life-cycle. To this extent, Tor Project published a Controller specification [54]. The document describes an implementation-specific protocol used by front-end applications wishing to control Tor locally. In other words, Tor Controller allows ordinary applications to start, stop, display logs, open and close circuits. For instance, the Tor Browser, which is a modified version of Firefox browser, implements the controller specification allowing users to swap the circuit pre-established by Tor for a given website.

The Controller specification is fully implemented in Java and Python using the Stem library[3]. Several partial implementations in C/C++ were found online, however, most of them do not implement the necessary commands to create, delete and control Tor circuits. To this end, we opted to implement, from scratch, only the vital commands to put TorK operational as the specification is quite extensive.

The protocol is stream text-based which can be implemented by TCP or Unix sockets. Usually, all messages from Tor Controller are replies to Tor clients' commands previously sent, except asynchronous

[3]`https://stem.torproject.org/`

event messages. An event consists on alerts about Tor behaviors, such as creation, deletion of circuits and streams. A Tor controller client can subscribe events by categories and receive them in the future.

Internally, Tor controller is implemented in two threads (Figure 5.4). The main thread is responsible to receive both events and command replies. The second thread aims at handling event alerts subscribed at startup. To efficiently handle controller messages, a ring buffer is used for each message type: replies and events. Whenever a new reply or event is received, the main thread adds it to the respective buffer. The second thread uses a producer-consumer scheme, gathering events from the buffer and invoking the handler to execute a proper action. The controller exposes functions capable of creation, closure and attach of streams which are used by the Bridge/Client Controller.

Streams are user opened connections that can be attached to given circuits (e.g. HTTP request). By default, Tor handles streams automatically. In other words, Tor automatically attaches streams to circuits. Nevertheless, Tor also allows controllers, in this case TorK, to manually handle streams. Since TorK needs to rein all aspects regarding a circuit life, TorK manually reroutes streams to the given circuit only when authorized per bridges as part of the protocol.

5.5 TorK Controller Interface (CLI)

Tor exposes a controller to tune Tor settings and receive events as described in the previous subsection. Using the same approach, TorK exposes to end users a text based stream protocol (CLI). The protocol grants choice on the k-min restriction (i.e. the minimum number of participants that must coexist in a k-circuit), choice whether a client intends to open or close a Tor circuit or remain chaff-only. The CLI is implemented as TCP and Unix socket, alike Tor Controller.

CLI is implemented as a simple socket server, receiving commands, passed them to Bridge/Client Controller handler which take a certain action depending whether is a TorK bridge or TorK client gateway, and processes a reply back to the user. CLI is used primarily by clients. Bridge CLI was used only to gather statistical data for evaluation purposes. For instance, the main TorK CLI commands available are:

- `set_k`: defines the k-min value provided by the client,
- `pset_k` and `jset_k`: grant clients the possibility to create a private k-circuit or join an existent private k-circuit, respectively,
- `ch enable / ch disable`: appeals bridges to open or close a Tor circuit,
- `shut`: triggers a controlled and coordinate shutdown between the bridge and the client.

5.6 Traffic Shaper

TorK employs a component capable of sending frame chunks at a specific rate – *traffic shaper*. Thus, TorK traffic shaper sends data to every connected client at a specific rate defined in milliseconds. Bridges

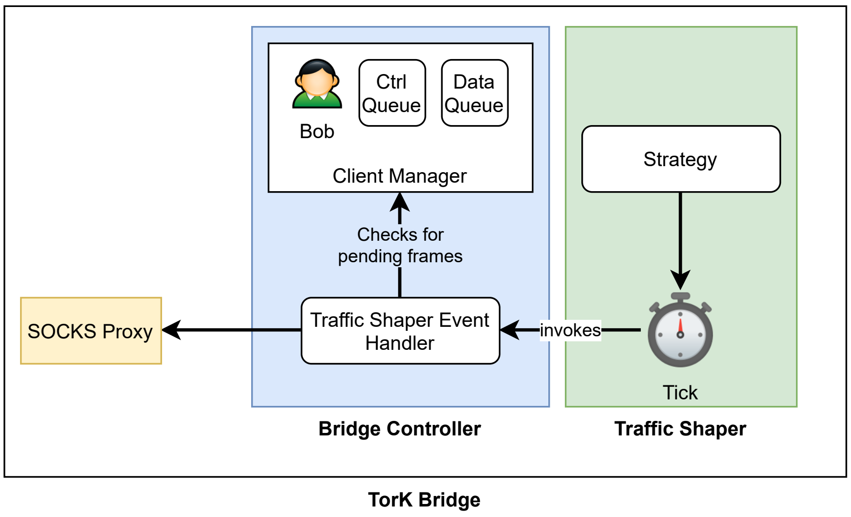


Figure 5.5: TorK traffic shaper.

can change the sending rate of the clients dynamically. The sending rate is calculated based on the number of clients currently connected and the size of TorK frame chunks.

Bridges maintain data and control outgoing queues per connected client. It aims at buffering outgoing data frames and control frames which are going to be processed in the next traffic shaper event. The traffic shaper assigns higher priority to control frames than data frames allowing control messages to be delivery at the next traffic shaper event. Under the same policy, no chaff frames are sent when at least one data frame remains buffered to be sent.

Upon a clock tick, an interruption handle is invoked from the Bridge Controller – a Traffic Shaper Event. This event takes care of sending chunks from buffered frames or chaff. Internally, the Traffic Shaper main function runs in a separated thread, which periodically executes the handler. The frequency of such tick is designed as the strategy. Several strategies were studied: i) *constant* rate defined at start by both bridges and clients and do not change during the TorK lifetime (e.g. hardcoded constant), ii) *exponential* whose value follows a exponential distribution and iii) *dynamic* assigned which can vary according to the number of connected users. The constant strategy proved to be very inefficient when the number of users is high (i.e. close to 50). The exponential strategy even though provides small variations suffers from the same issue as the constant. Thus, the dynamic approach proved to be the best solution which optimizes the rate according to the number of connected users, allowing higher rates when the number of clients is lower.

The Traffic Shaper state can be described by a state machine, and is depicted on Figure 5.6. Bridges start traffic shaper in *Idle* state whereas client gateways start in *On*. The *Idle* state prevents the TS from calling the handler every tick when there are no users connected to the bridge, and then wasting CPU cycles to unnecessarily process the handler. As soon as there are users connected, the TS becomes *On*, executing the handler every clock tick, as usual. In order to terminate the TS component cleanly, the TS wait for a specific TorK event. Upon the event triggered, the TS goes into *Shutting* mode and starts all the actions to properly close both threads. Finally, when the shutting process terminates TS is *Off*, which TorK can successfully exit, since all threads and TS resources are properly closed.

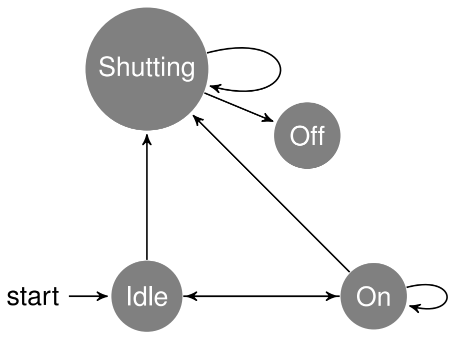


Figure 5.6: Traffic Shaper state machine.

5.7 Bridge/Client Controller

The Bridge/Client Controller acts as a central block of TorK. This controller defines functions handlers to process different Tor controller events, CLI events and handles to process every frame type. For instance, whenever data is available at the Socks Proxy, a Bridge/Client Controller is invoked to take action on the data, i.e. convert data into TorK's custom message format (see Section 5.7.1). It implements the TorK Protocol (Section 4.2.1).

Client Controller also handles CLI commands and Tor events. For instance, whenever a clients performs a request to open a circuit, the Controller ensures that can safely carry out the operation, by requesting bridge's permission. Additionally, Bridge Controller maintains a client bookkeeping – Client Manager. The bookkeeping helps bridges contains an up-to-date record containing connected clients, k-restrictions, internal states and the reception queues/markers. The latest aim at protecting bridges from active network attacks, and will be discussed in Section 4.3.2.

Bridge/Client Controllers also implement a Traffic Shaping Handler to perform action signaled by the Traffic Shaper. E.g, bridges respond to TS events by iterating the list of connected clients and their queues to perform one of the following actions: i) If client's queues (control and data) are not empty, a chunk from the frame at the top of the queue is popped and sent. A frame is only popped from the queue after sending the last chunk. ii) If client's queues are empty, a chaff frame containing one chunk and created at the initialization, is sent to the client. Clients proceed to TS events using the same approach.

Finally, bridge controllers implement the Bandwidth Control (described in Section 4.3.2). To this end, Bridge Controllers use a reception queue to store stalled data frames received from every client and a marker to indicate whether a client received a CHAFF or CTRL frame. If all clients have non empty reception queues or the markers are set, one or more DATA frames are dispatched to the Tor Network, resetting the marker. CHAFF and CTRL messages are always processed upon reception as they do not carry Tor cells and thus processing such frames do not result in sending data to Tor.

5.7.1 TorK Frames

TorK receives Tor cell traffic and encodes it into specific TorK cells – *frames*. Figure 5.7 depicts the internal format of TorK's frames. Each frame is divided into smaller slices called *chunks*. Each chunk shares the same size and properties, apart from the first one. All chunks except the first one can carry

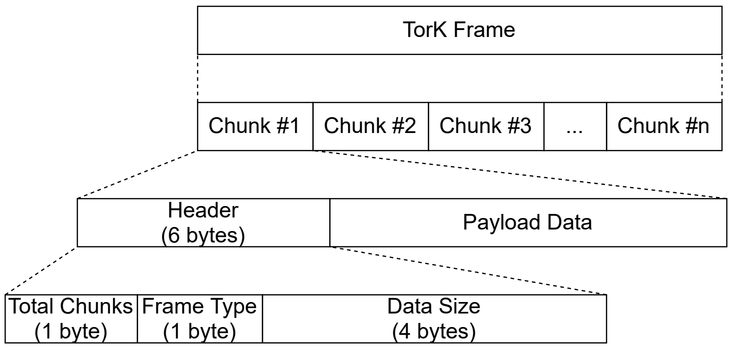


Figure 5.7: Format of TorK frames.

data entirely. In other words, the entire chunk space can be taken to fill with data. The first chunk contains a frame header which occupies 6 bytes, thus the first chunk carries less 6 data bytes. The first chunk is the only chunk carrying a header. The header consists on three fields: total number of used chunks, frame type - data, chaff or control which will be discussed in Section 4.2.1 - and data size corresponding to the size, in bytes, of the data payload.

Each frame has a fixed size and a maximum number of chunks. Such design choice provides efficiency by preventing Tor cells fragmentation over multiple TorK frames. In other words, this design allows to have TorK frames large enough to contain one or several Tor cells without fragmentation while sending only the necessary data (chunks) minimizing the sending of unnecessary bytes. For instance, a solution without considering chunks, where a frame were essentially the first chunk without considering the size of the Tor cells, would result in a large unnecessary amount of data. We argue that the selected solution, considering chunks, considers both the network efficiency and performance efficiency (preventing fragmentation).

One of the frames implementation challenges faced was how to handle the frame memory space efficiently. Essentially bridges will handle with multiple requests from multiple clients maneuvering several frames at once. Several solutions were studied. The first approach consisted on frames' allocation on demand, based on the number of connected clients and their needs. Such solution reveal itself as a performance burden due to expensiveness of dynamic allocation and also memory fragmentation. Other solution was allocating memory once for frames during the TorK startup phase. Essentially, bridges alloc a number of fixed frames from the start. Internally, frames are organized by a frame pool, thus they are used and reused without dynamic allocation, without any memory operations apart from the initial allocation and deallocation when exiting.

5.7.2 Control Frames

Apart from chaff and data frames, TorK uses a special type of frames – control frames. Such frames aimed at implementing the TorK protocol (section 4.2.1). Control frames instruct TorK to construct or tear-down circuits, and are depicted in Figure 5.8. The majority of control frames uses 2 bytes containing only a header. The header contains the total chunks and the frame type similarly to the data and chaff frames. However, control frames do not use any payload data. Some TorK commands such as `Hello`

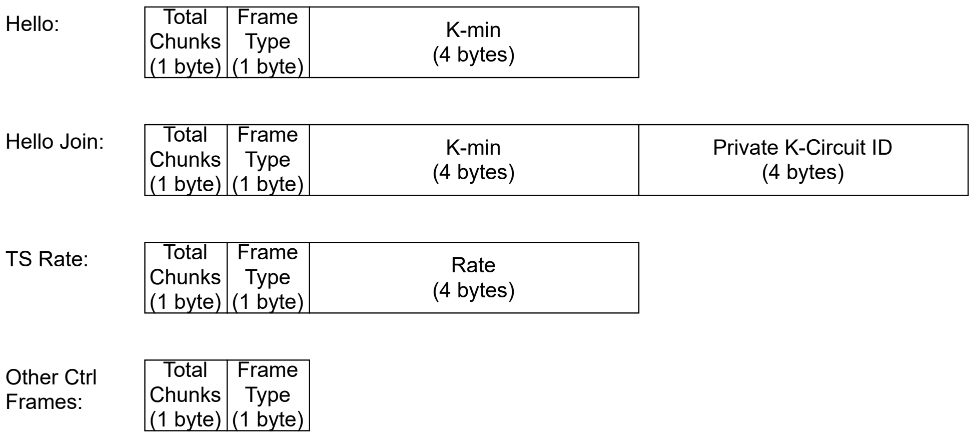


Figure 5.8: Format of TorK control frames.

and `Hello Join` which allows users to define their k-min to join a public k-circuit or to join an already existent private k-circuit, may use up to 10 bytes. The `TS Rate` allows bridges to increase / decrease the client gateways TS rate dynamically. Likewise chaff and data frames, padding is added to control frames filling the chunk up to the defined fixed size, preserving the same size properties across all frame types.

5.7.3 Frame Transmission

TorK's indistinguishable channel encloses a Traffic Shaping handler function. Such function is responsible for sending fixed sized TorK traffic – called *frames*. The size of the frames should be set to a multiple of the Tor cell sizes to avoid split Tor cells across multiple frames. The frames transmission rate is also configured to provide a target throughput.

Whenever Tor sends a cell to TorK, the cell is enclosed in one or several chunks within a frame, or even several frames, which are later sent. In order to maintain chunks fixed size, padding is added when the content length is smaller than the chunk size. Whenever no Tor traffic is available, TorK uses a dummy frame, containing one *chaff* chunk, respecting the same size properties. Such frame is generated at the beginning of TorK execution and is discarded upon reception at the bridge. The purpose of this type of frame is to ensure that packet size and timing properties do not leak traffic bursts or identifiers.

The channel hides packet sizes and timing properties which could be explored by network adversaries. Even though Tor cells are encrypted in several layers (one for each relay), TorK also assures a secure TLS connection between clients and bridges using the OpenSSL library. To this end, an eavesdropper cannot identify the type of a given frames (e.g. control, chaff or data) by inspecting the Frame Type byte located in the header. Otherwise an attacker can not only identify whose frames are chaff or data but also read TorK control commands (e.g. identify the client that open a given circuit).

5.8 Shielding the TorK Bridge with SGX Enclaves

The solutions presented so far strengthen TorK's resilience against powerful network attackers able to delay or drop arbitrary packets. Firstly, it is considered a scenario where an attacker can control all

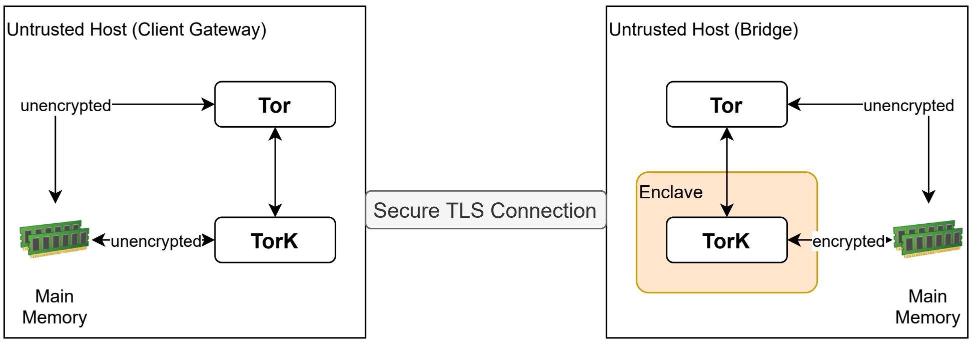


Figure 5.9: TorK Bridge running inside an enclave.

network hops between users and their destinations, thus some TorK leverages protection mechanisms against intentional delays as the bandwidth protection described previously. Such protections hold under the assumption that all TorK entities are trustworthy. However, is reasonable to assume that a powerful network attacker, may wittingly deploy malicious TorK clients or bridges easily. To hardening TorK against crook clients and bridges we endow additional measures using secure *enclaves* on top of Intel® Software Guard Extensions, also known as Intel® SGX.

Be able to protect against malicious clients and bridges translates into running TorK instances on top of non trusted operating systems shared among different processes and users. One of the most important aspects is to prevent the system administrator from accessing TorK memory variables. Such memory inspection can leak information about the number of real users vs dummy/chaff users and reveal encryption keys used to protect in-transit data. One core motivation behind SGX is to protect data from unauthorized access and modification by other process, even with higher privileges.

Notwithstanding protecting memory and encryption keys, a malicious client can deliberately change TorK source code, recompile it and connect to a trustworthy bridge. Such changes put himself at risk – *which is not relevant since it is malicious* – but also put the remaining bridge users at stake. Therefore, it is crucial to provide a mechanism to ensure that all connected clients are running trustworthy copies of TorK. The most popular solution to verify software integrity is SHA256 hash which is usually signed by key, using Pretty Good Privacy (PGP). Such approach are used in the context of file sharing and email to preserve its integrity and confidentiality, respectively. SGX provides a mechanism of remote attestation, giving the ability of both clients and bridge to attest that the other part is running a valid version of TorK inside a secure enclave. To this extent, TorK would be able to support remote attestation of bridges, to reassure clients that bridges are trustworthy. In the opposite way, remote attestation of clients allow bridges to discard invalid clients ensuring K-Circuit of trustworthy clients.

To provide confidential computing, we opted to use SCONE [56] confidential computing platform. SCONE allows ordinary Docker containers to use SGX trusted execution on top of Intel CPUs. The core idea lies in protecting the code and the data of Docker containers, in this case TorK, such that no user, even the root users, be able to access them. SCONE simplifies the process of encryption/decryption main memory data, input and outputs transparently.

To support SGX, no significant changes were made as SCONE provides a standard C library interface. A Docker container image was created and TorK was recompiled using a C++ cross-compiler [56] with SCONE support. It was necessary to compile OpenSSL library from source and link all dependencies statically to TorK, as SCONE does not support shared libraries by design [56], ensuring that all enclave code is verified by SGX at the creation of the enclave.

Figure 5.9 depicts the components running inside a SGX enclave. We opted for running only TorK instance inside an enclave, leaving the Tor process running outside of a secure enclave. Such design choice aligns with current constraints of space regarding the Enclave Page Cache (EPC). Typically, EPC size is up to 128 MB, having a high number of processes accessing memory pages outside of EPC, causes a higher number of cache misses and thus, pages must be read and decrypted from main memory, adding a considerable overhead.

Summary

This chapter has described the implementation details of TorK prototype. The technical setup operation, the indistinguishable channel along with the TorK frames allow setting up a resilient channel against correlation attacks. The traffic shaper is a core component of the channel which controls the rate of outbound traffic and can be adjusted using different strategies. Assuring an indistinguishable channel is not enough to withstand a powerful attacker, thus TorK employs the use of a protocol aim at synchronizing actions among clients and bridges taking into account network disruptions. Finally, the chapter ends by hardening TorK main fragility: clients and bridge trustworthiness, by using hardware support to run TorK in a shielded environment, withstanding against a local attacker capable of inspecting TorK memory space.

Chapter 6

Evaluation

This chapter describes the evaluation of our system. We start by explaining our methodology (Section 6.1). Then, we study and analyze TorK's performance under different settings (Section 6.2), and show TorK's effectiveness in generating indistinguishable k-anonymous circuits (Section 6.3) even in the presence of robust against active network attacks aimed at compromising k-anonymity. Next, a comprehensive study respecting hardware resources such as network and CPU usage is presented (Section 6.4). Lastly, we introduce a mechanism based on Intel SGX that can ensure the correct behavior of TorK nodes when the execution runtime is compromised by an attacker (Section 6.5).

6.1 Methodology

This section describes our evaluation methodology. First, we describe the goals and approach of our evaluation. Then, we present the laboratory testbed that we designed for performing our experiments with TorK and the metrics used to assess the quality of our solution. Lastly, we explain our choice of TorK's traffic shaping parameters for our experiments.

6.1.1 Evaluation Goals and Approach

Our main evaluation goals are twofold: 1) assess the performance of our TorK prototype, and 2) measure its resistance against multiple traffic analysis attacks. In particular, regarding TorK's resistance against traffic analysis attacks, we wish to assess whether it is possible a) to identify which flows between TorK clients and bridges transport useful data vs those that only carry chaff traffic, and b) to correlate the useful data flows produced by clients with the outbound flows from the bridge towards Tor.

Recent literature aimed at distinguishing encrypted traffic flows has focused on the analysis of packet length and inter-packet arrival times. Typically, existing approaches employ classifiers based in machine-learning, such as those based in decision-trees (e.g., Random Forests [57] or XGBoost [51]). Akin to earlier traffic analysis attacks over encrypted network streams, we leverage the XGBoost [51] classifier to assess TorK's indistinguishability properties. For this classifier, we leverage features based on multiple statistics over the packet length and inter arrival times, such as burst behaviors and high-order statistics (e.g., kurtosis), and quantized packet length distributions. We train and test the classifier through 10-fold stratified cross validation.

6.1.2 Experimental Testbed

Our laboratory testbed is composed of a Tor bridge and 50 clients, where each node runs an instance of TorK (leveraging Tor v0.4.2.8). Our nodes are executed within Docker containers, provisioned either with 2 vCPU and 2GB of RAM in the case of the bridge node, or 1 vCPU and 1GB RAM for client nodes. The bridge and client nodes run in separate physical hosts equipped with 2 Intel Xeon CPU E5506 vCPUs.

To select a reasonable number of users to test TorK we estimated the amount of daily connected users in Tor bridges based on monthly statistics provided in the literature [9]. Thus, 50 clients represent an average of daily users in Tor bridges. Each client was configured to use a specific Tor circuit throughout our experiments. We manually selected all relays comprising each Tor circuit based on Tor's *TopRelays* list [58], and chose different nodes within Europe. Further, for comparing the throughput and latency obtained by our vanilla prototype of TorK against its hardened version when using Intel SGX, we resort to Docker containers to perform our tests under multiple resource constraints.

For conducting the above experiment, we build a representative dataset of multiple connections between clients and a bridge, and, respectively, between the bridge and the Tor network. For case a), we select half of our clients to generate chaff traffic towards the bridge, while the remaining 25 clients will fetch files with sizes, ranging from 32 MB to 64 MB, hosted in a private server, over TorK, repeating this process 100 times (for a total of 5000 samples). For case b), we use all 50 clients to synchronously fetch files, repeating this process 100 times (for a total of 5000 samples). For both experiments, we capture traffic in/outbound the bridge during 15 seconds.

6.1.3 Metrics

We adopt a set of metrics for evaluating TorK's performance and resistance against traffic correlation:

Performance Metrics: To measure performance we studied the throughput and latency under different chunk sizes: 536 B, 1050 B, 2078 B and 4134 B. These values allow chunks to contain up to 1, 2, 4 and 8 Tor cells, respectively. The traffic shaper rate function was modified to maintain the same minimum and maximum throughput. Thus, the rate function selected was $Rate(t, n) = 10 \times t \times n$ µs, where t is the number of maximum Tor cells that fit one chunk and n the number of connected clients. Such function offers a theoretical throughput of 8.5 Mbps to the 50 connected clients regardless the frame chunk size.

Security Metrics: We leverage the following metrics to assess TorK's k-circuit indistinguishability: *true positive rate* (TPR), *false positive rate* (FPR), and the *area under the ROC curve* (AUC). For case a), the TPR measures the fraction of TorK flows that are correctly identified as carrying useful data, while the FPR measures the fraction of chaff traffic flows that are erroneously classified as useful traffic. For case b), the TPR measures the fraction of client-bridge and bridge-Tor pairs that are correctly correlated, whereas the FPR measures the fraction of uncorrelated pairs that are deemed as correlated. The Receiver Operating Characteristic (ROC) curve plots the TPR against the FPR for the different possible

TCP Data Length	# Packets	Percentage of packets
33	2	0.43 %
80	2	0.43 %
317	2	0.43 %
536	401	86.05 %
1050	54	11.59 %
1072	1	0.21 %
1564	3	0.64 %
2078	1	0.21 %
	466	

Table 6.1: Distribution of TCP length packets carrying Tor cells.

cutout points for classifiers possessing adjustable internal thresholds. The AUC summarizes this trade-off. Note that an AUC of 0.5 is equivalent to random guessing.

6.1.4 Choosing TorK's Traffic Shaping Parameters

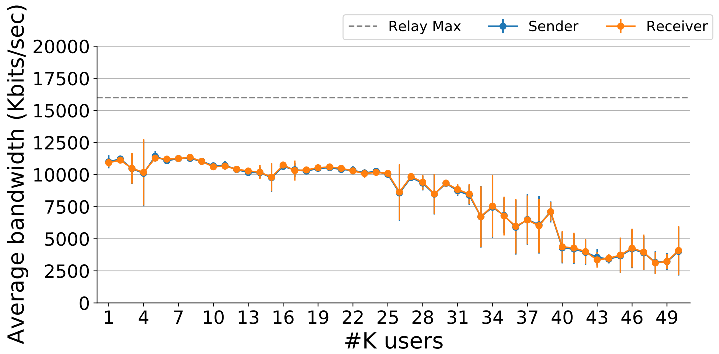
TorK can be configured to create an unobservable channel based on either the frames size and the interval rate. Ideally we intend to achieve the largest throughput targeting a minimal overhead. Achieving the best values for both variables (frame size and interval rate) requires to understand what are the most common size distributions of Tor circuit cells. This is particular relevant since Tor multiplexes multiple cells over the same TCP stream along the same circuit [1]. Being a pluggable transport that receives Tor cell data from Tor, TorK should be configured to minimize overhead. Such overhead may occurs when the frame size is too small to hold a single Tor cell and then creating fragmentation. Tor used 512 bytes cell, and recently moved to 514 bytes cell [59] since old cells caused relays to run out of circuit IDs.

By dissecting a network trace it was possible to discover that the TCP data length of a Tor cell is 536 bytes, considering the TCP and TLS headers. The Table 6.1 presents the distribution of TCP data length of captured Tor cells. We can see that most of TCP packets (86 %) received by TorK are 536 bytes, which corresponds to a single Tor cell. A detailed analysis of the collected traces revealed that the distribution of packet sizes follows a linear function such as: $Bytes(n) = 514n + 22$. However, 12 % of captured traces indicate that two Tor cells share the same TCP packet. $Bytes(2) = 514 \times 2 + 22 = 1050$ Based on the results of Table 6.1 we can find an optimal range for TorK frames size. The value should be greater or equal than 536 bytes, to hold a single Tor cell without any fragmentation, or up to 2078 bytes to hold up to four Tor cells within each chunk. Specifying a value greater than 2078 bytes may provide less Tor cells fragmentation but, in turn, would increase the number of unnecessary bits to send.

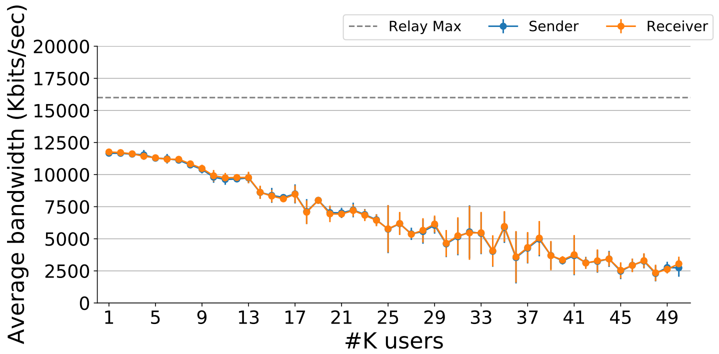
6.2 Performance

6.2.1 Throughput

Using a fixed size frame allow us to provide a indistinguishable channel but also will incur in an overhead. To quantify the performance of the system an experiment was conducted to measure the covert channel



(a) Clients receiving chaff.



(b) Clients receiving data.

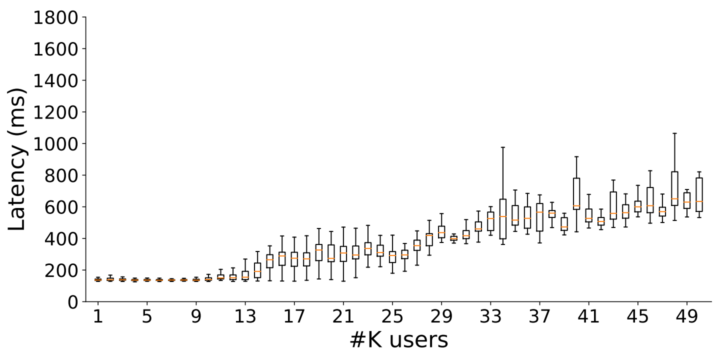
Figure 6.1: TorK throughput for the baseline (536 bytes chunk) with an increasing number of clients.

throughput. For the baseline experiment, we set TorK to use 536 byte frames, each frame containing up to 2 chunks, over a dynamic rate interval $Rate(n) = n \times 10$ µs, where n is the number of connected clients. Which lead to a maximum throughput of, approximately, 8.5 Mbps with 50 connected clients. To measure the channel throughput, we used IPERF3[1]. The client runs an instance through TorK and consequently through a specific 3-relay path performing the measure to the server. Figure 6.1(a) depicts the achieved throughput based on the average bandwidth of 5 runs according to the increasing number of connected users K receiving chaff data. Instead, Figure 6.1(b) depicts the achieved throughput according to the increasing number of connected users K receiving actual data through Tor.

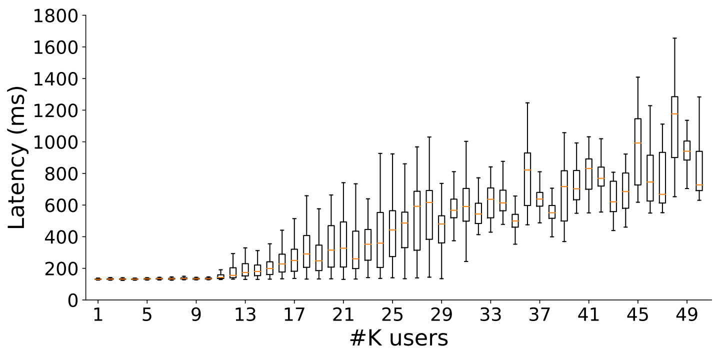
For testing purposes all 5 runs used the same relay configuration. Even though relays advertise bandwidth as part of the Tor consensus, relays may also impose bandwidth restrictions per user/relay connection using the TORRC options: PERCONNBWRATE, PERCONNBWBURST, RELAYBANDWIDTHRATE and RELAYBANDWIDTHBURST which use token buckets to rate the bandwidth of client or relay data evenly to every connected client/relay discouraging greedy users. Unlike advertised bandwidth, per connection rates are not public, hence we conducted an throughput experiment (in 5 runs) without TorK under the same relays configuration to measure the actual per client throughput of the testing relays. On average, under this testing setup, we observe a 16 Mbps throughput (denoted as a gray line).

Our results show that the one client can achieve up to, approximately, 12 Mbps of 16 Mbps maximum achieved by these relays when using TorK. With a higher number of connected clients, TorK decreases

[1] `https://iperf.fr/`



(a) Clients receiving chaff.



(b) Clients receiving data.

Figure 6.2: Baseline configuration latency observed by number of bridge clients.

the rate and thus reducing the throughput. This reduction is more critical when most clients are, in fact, receiving useful data instead of chaff. This behavior is depicted in Figure 6.1 and is inline with TorK's expected behavior. Processing chaff frames is less burdensome than data frames since the former are discarded upon reception and the latter require receiving all chunks and deliver the content to Tor.

6.2.2 Latency

To measure the latency of the channel, we conducted a RTT measure over the channel. The use of ICMP to measure the RTT, through PING tool over the Tor network was not possible since Tor does not route any other packets apart from TCP. Thus HTTPING[2] tool was used. This tool performs a HTTPS request and measures the time it takes to receive the first byte of the header – header time to first byte.

Figure 6.2(a) depicts the average latency of 5 runs according to the increasing number of connected users K receiving chaff data. Instead, Figure 6.2(b) depicts the achieved throughput according to the increasing number of connected users K receiving actual data through Tor. An increase in the number of connected users causes the latency to increase. Anew data frames causes slightly higher increases in latency than chaff frames. It is important to notice that the bandwidth control measure is also in place (described in more detail in Section 6.3.4). Therefore, the delivering process of data frames' content (to the Tor Network) may be intentionally delayed as reactive measure to active attacks.

[2]https://www.vanheusden.com/httping/

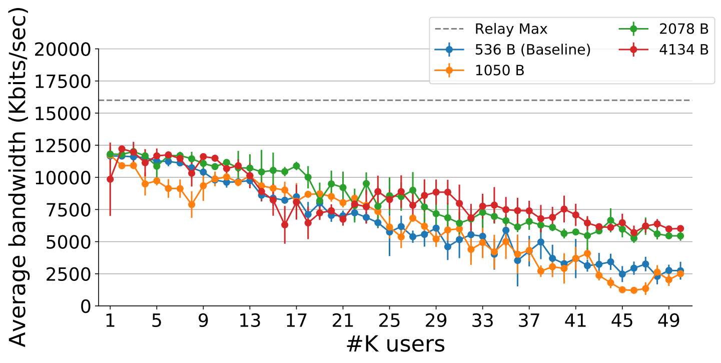


Figure 6.3: Observed throughput with different chunk sizes. All clients are receiving data.

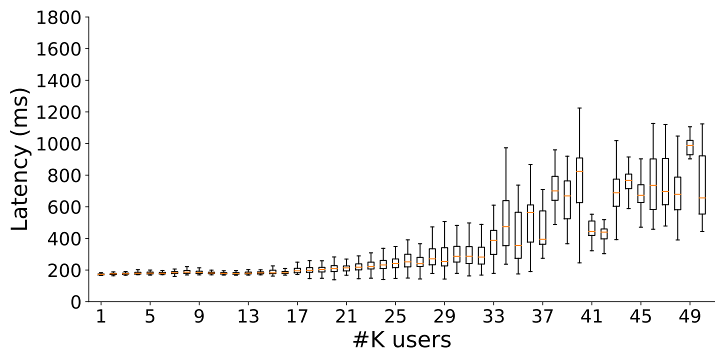
6.2.3 Varying Traffic Shaping Parameters

Our baseline experiment uses 536 B chunk size which can carry a single Tor cell within each chunk. To study how the chunk size affects the throughput and latency, the baseline experiments were repeated under different chunk sizes: 1050 B, 2078 B and 4134 B. These values allow chunks to contain up to 2, 4 and 8 Tor cells, respectively. The traffic shaper rate function was modified to maintain the same minimum and maximum throughput. In other words, the TS rate minimum and maximum values for each chunk size are: (10 µs,500 µs), (20 µs,1000 µs), (40 µs,2000 µs) and (80 µs,4000 µs). Figure 6.3 depicts the throughput results with different chunk sizes. In theory, the throughput should remain approximately the same for each configuration, which is 8.5 Mbps. However it is possible to observe that for the same number of 50 connected clients, having a higher chunk sizes allows TorK to approximate the throughput to the maximum theoretical values. Since the maximum theoretical throughput remains the same across all chunk sizes, 2078 B / 4134 B turns to be more efficient than the baseline chunk size (536 B).

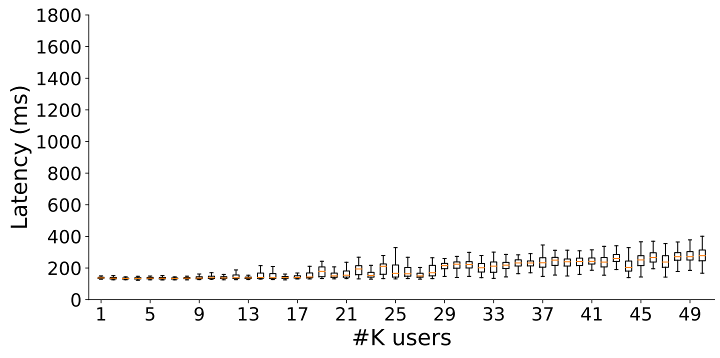
Baseline latency results shows increased latency values for higher number of participants as expected, having higher incidence when $\#K > 13$. Comparing baseline latency results (Figure 6.2) against the results depicted in Figure 6.4(a), the same behavior occurs only with $\#K > 25$. Such behavior may occur due to fragmentation of several multiplexed Tor cells inside a single TorK frame chunk in combination with higher CPU load at the bridge. If one or several Tor cells are scattered over several chunks, it will require multiple traffic shaping iterations to send that Tor cell. Additionally to fragmentation, a high number of participants will cause the latency is to increase due to the additional overhead at bridges. However, when the chunk size comprises up to 4 Tor cells (Figure 6.4(b)), there is almost no large latency fluctuation even with 50 connected clients. Such results show that higher the chunk is, the less prone TorK is to Tor cell fragmentation.

6.3 Resistance against Traffic Analysis

This section evaluates TorK's ability to create a indistinguishable channel in the sense of protecting the identity of the real sender in a group of K members. Specifically it addresses how TorK resists against a number of traffic analysis attacks. Chiefly, it studies passive network attacks where an attacker



(a) 1050 B chunk (up to 2 Tor cells per chunk).



(b) 2078 B chunk (up to 4 Tor cells per chunk).

Figure 6.4: Throughput and latency by number of clients when using different chunk sizes; clients are receiving data.

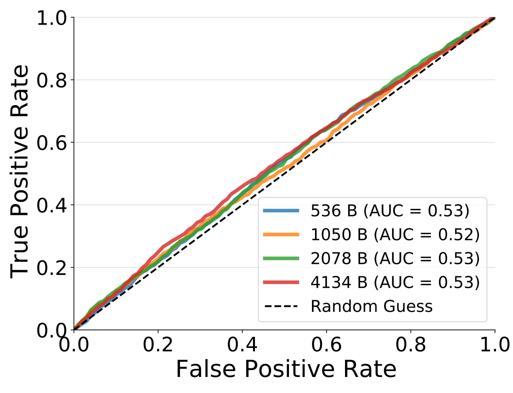
may, solely by inspection, glean statistical information (e.g. packet sizes, timing properties), and use machine learning algorithms to find meaningful data patterns. Active attacks, where an attacker wields interference such as packet drops and delays observing how TorK responds, were also studied.

6.3.1 Distinguishing between Chaff and Useful Data

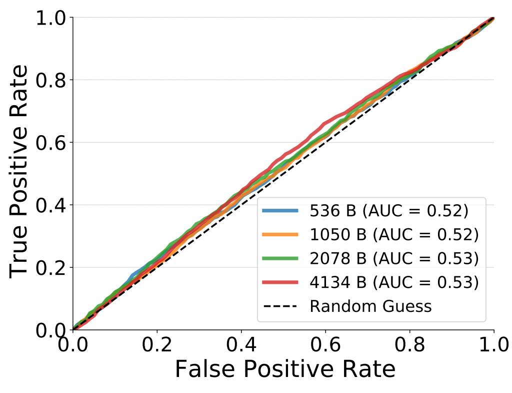
First, we study if TorK bridges disclose the identity of clients who are sending or receiving Tor traffic, or dummy traffic based on inspection of their network traces. If such scenarios occur it would be trivial for a network attacker to identify the origin of certain Tor traffic cells, and consequently reveal the client's identity. To study this scenario, we set two clients and one bridge. The first client downloads a file using the Tor network where the second client establishes a chaff only connection to the bridge.

We conducted a first experiment to verify if regular Tor traffic and chaff traffic can be discriminated based on the inspection of network traces. A second experiment tries to link a given bridge incoming flow with its given outgoing flow, in order to verify if it is possible to correlate the two flows based on their properties. The goal is to train the classifier with pairs of both match and not match labels and try to predict if a given bridge incoming flow is tied to a certain Tor circuit flow.

Figure 6.5(a) depicts the XGBoost ROC curves using summary statistics under differ chunk sizes and traffic shaping rates configuration. The summary statistics include the minimum, maximum, mean, standard deviation, percentiles, kurtosis and skew. Kurtosis measures the distribution of the data relative



(a) Summary Statistics



(b) Packet Length

Figure 6.5: Performance of the XGBoost classifier when attempting to distinguish between k-circuits carrying real Tor data and chaff.

to a normal distribution and skew measures the symmetry of the packet length.

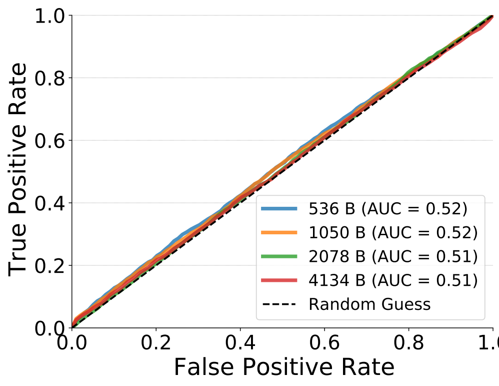
Figure 6.5(b) depicts the XGBoost ROC curve using only the packet length (PL) statistics. We plotted the True Positive Rate (TPR) which measures the fraction of correctly identified TorK flows while the False Positive Rate (FPR) measures the fraction of TorK flows that were wrongly classified (e.g. a chaff client flow is classified as Tor traffic). A network adversary will aim at obtaining a high TPR while maintaining a low FPR. The PL encloses a frequency distribution table, represented in buckets of packet lengths. The PL captures then the frequency of a given packet length during the experiment. Unlike many of machine learning problems we aim at designing the system to obtaining the worst accuracy classification possible. Such results would indicate that a classifier is no better at labeling pairs (chaff or Tor traffic) than the random guessing. In other words, the classifier brings no advantage point to an attacker aimed at inspecting packet length or timing properties to reveal what clients are sending chaff or real Tor traffic. As conveyed by both figures, the AUC of the classifier for summary statistics and PL is very close to 50%, i.e., to the AUC of random guesses. We then see that TorK is very effective at making the traffic that carries chaff and the traffic that sends useful data indistinguishable from one another.

6.3.2 Matching TorK clients with Outgoing Bridge Traffic

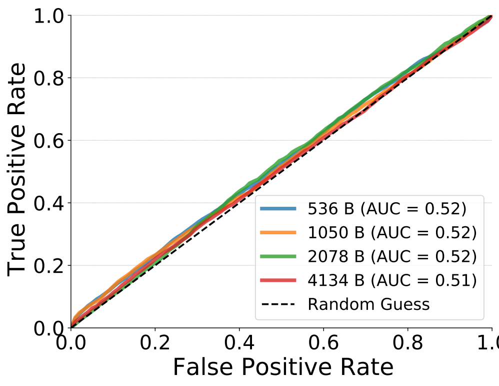
A second study consists on setup 50 clients and 50 different Tor circuits. Instead of training chaff vs. non chaff traffic, pairs of ingress and egress flow are trained. Therefore, the goal of the classifier is to find a match between TorK traffic and their corresponding Tor circuit.

Observing the constant results (Figure 6.6), it can be concluded that it is not accurately possible for an attacker to correlate a given TorK flow with the respective Tor circuit solely based on packet inspection. We plotted the True Positive Rate (TPR) which measures the fraction of correctly identified pairs while the False Positive Rate (FPR) measures the fraction of pairs that were wrongly classified (e.g. a chaff client flow is classified as Tor traffic).

It is possible to conclude that an attacker cannot measure patterns that allow differentiation in chaff only and non chaff traffic nor differentiation between different Tor circuits. Thus, finding a correlation



(a) Summary Statistics



(b) Packet Length

Figure 6.6: Performance of the XGBoost classifier when matching a given TorK flow with the corresponding Tor circuit.

based on TorK and Tor summary statistics (Figure 6.6(a)) or using packet length frequency distribution (Figure 6.6(b)) brings no better results to an attacker than random guessing.

6.3.3 Leveraging Circuit Destruction Cues

An attacker may notice a client's identity after a re-utilization of a Tor circuit. In a k-group of two users, when one user leaves and, later, another one joins, the oldest client must use a new Tor circuit. Continuing using the oldest circuit reveals its identity. A watchful attacker knows that a specific circuit was associated with one of those two users before one decides to leave. Later, when a new user joins the attacker can correlate them with their Tor circuit.

In order to test such possibility we conducted several tests. Consider a scenario with three users joining in a k-circuit with $k = 3$. They connect to the bridge at different times requesting a circuit opening, first client one, then two, and lastly the third client. Since all three clients define k as three, it is only possible to construct circuits after the join of the third client. To obtain accurate data about how many bytes and when they are received or send by clients and bridges, a telemetry data was added to TorK for the purpose of statistical gathering. Given the above scenario, without any network limitation, bytes statistics are depicted in Figure 6.7.

At the initial instance only the bridge is alive. Around the instance of 10 seconds, client 1 joins the network. A joined client implies that the client must be at the *Connected* state (see Figure 4.5), therefore after defining their $k - min$. Twelve seconds after, at about instance 22s, client 2 joins and later, client 3 at instance 36s. At this point, both client 1 and 2 have authorization to create their circuits from the bridge, since there are at least three connected clients, as requested by both client 1 and 2. Around the instance 43s, client 3 who is playing chaff-only up to this point, decides to enter the Tor network. Bridge authorizes all of them and, according to the protocol (see Section 4.2.1), client 1 and 2 must change circuits. At the instances 55s, 63s and 67s, all three clients, respectively, start downloading a file from a private server. Around the instance 102s, client 1 download finishes, which translates in a decrease of bytes received from Tor network to the bridge.

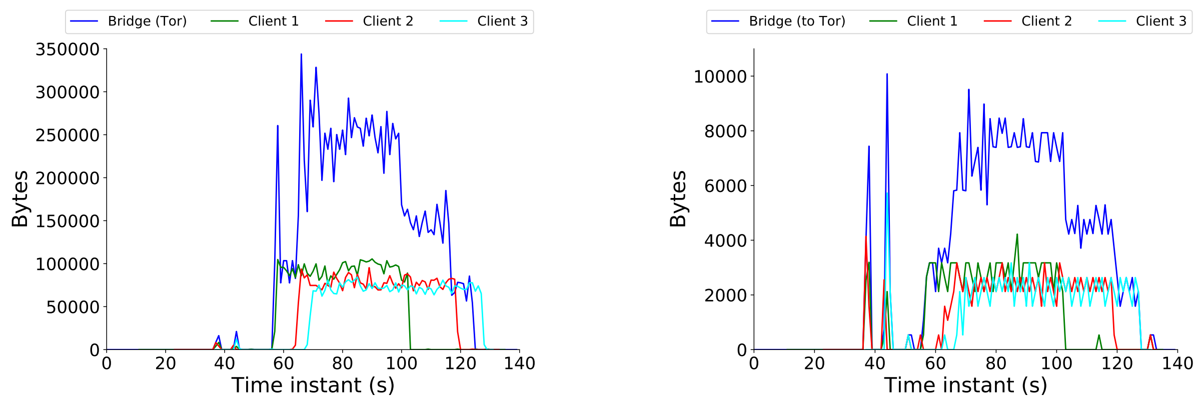


Figure 6.7: Bytes received (1) and send (2) by each client and the bridge during a regular scenario.

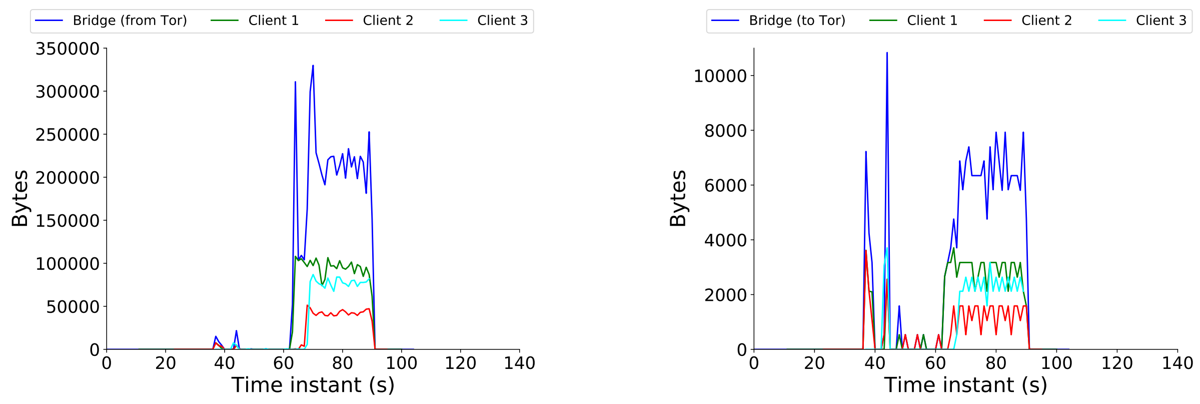


Figure 6.8: Bytes received (1) / sent (2) by clients & bridge when a client abruptly closes the connection.

However, a more interesting scenario is when, during the download, client 3 abruptly closes its connection. Such situation is depicted in Figure 6.8. Recall that, in the moment client 3 leaves, client 1 and 2 restriction became broken, i.e. there are no longer at least three clients connected. Around the second 90, client 3 abruptly closes the connection to the bridge. Client 1 and client 2 circuit's are destructed which matches the zero bytes received and sent from/to Tor right after the 90s instance. Both remaining clients do not receive or send any further traffic.

6.3.4 Bandwidth Control

To test a scenario where an attacker deliberately drops client's packets in order to disclose the respective Tor circuit an experiment was conducted. Two clients, forming a k-circuit with $k = 2$, will download the same file over the Tor network. During the process, at a specific instance, all packets to and from client 2 will be dropped during a limited period. It was used the Traffic Control – TC – Linux command to instruct the kernel queue discipline and emulate a total packet loss. Such situation enhances attacker's ability to study side effects and therefore, use it as a weapon to identify users.

Figure 6.9 depicts the bytes statistics without (left) and with (right) bandwidth control protection. At the instance 51, in the left example, all packets from client 2 are dropped. No bandwidth protection is employed, thus bridge continues to receive and sent traffic to client 1. After applying the bandwidth

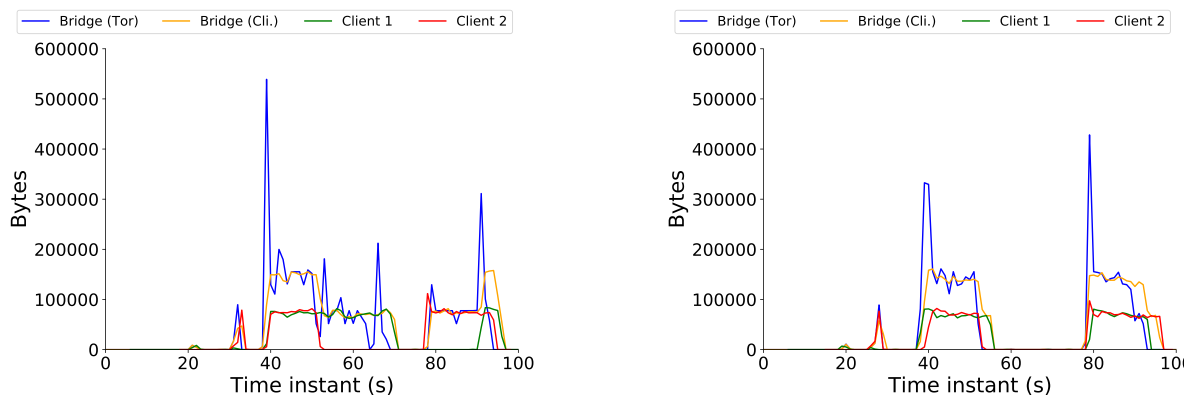


Figure 6.9: Bytes received by each client when dropping all packets from client 2, without (left) and with (right) bandwidth protection.

protection measure (right), the same dropping behavior on client 2 causes the bridge to stall all traffic to client 1. Around instance 80 in both examples, client 2 packet's drop is lifted, thus client 2 resumes the download and starts receiving traffic. The bandwidth protection mechanism proves to be very useful even in cases where malicious client deliberately change the software. For instance, the attacker can recompile from source, misrepresenting the traffic shaper by omitting the send of chaff frames.

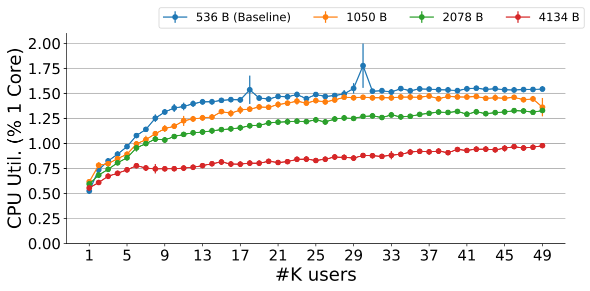
6.4 Resource Utilization

We choose to design TorK to offer additional protection against correlation attacks without harming the performance significantly. However, it will costs slightly increase in network utilization. This section studies and analyzes how much traffic, processing power and memory is required/generated by bridges and clients under different configurations.

6.4.1 CPU Usage

Figure 6.10 depicts the bridge CPU utilization in percentage of one core utilization. All experiments were conducted using the hardware limitations of the baseline experiment. Thus, bridges can use up to 2 cores under a 2GB As the chunk size increases CPU utilization decreases due to the TS rate value.

These results show the difference between processing chaff frames and data frames under different chunk sizes. Since chaff frames do not need any CPU processing apart from receiving and discarding them, processing chaff frames is a lightweight task in comparison with data frame processing. Figure 6.10(a) shows that smaller chunk sizes require more CPU processing. Since the rate is higher with lower chunk sizes, a baseline configuration requires more CPU processing than other configurations. Observing the bridge CPU usage when the clients are receiving data (Figure 6.10(b)) shows the same situation as the chaff processing regarding chunk sizes. However, since data frames require more processing than simply discarding frames, we observe a increasing trend across all chunk sizes.



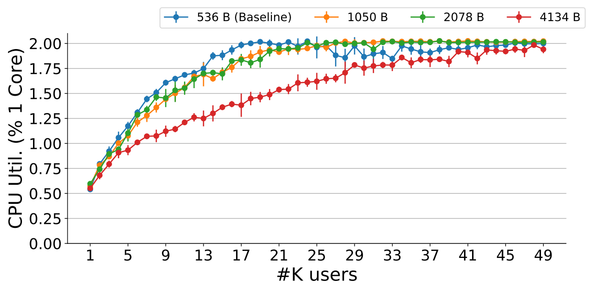
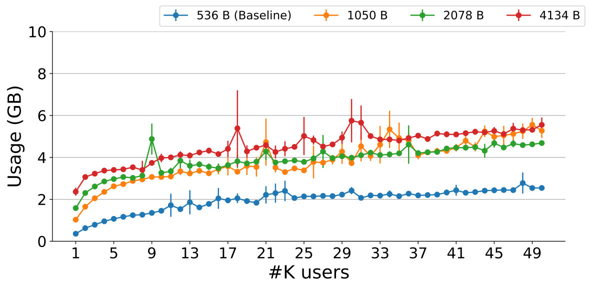
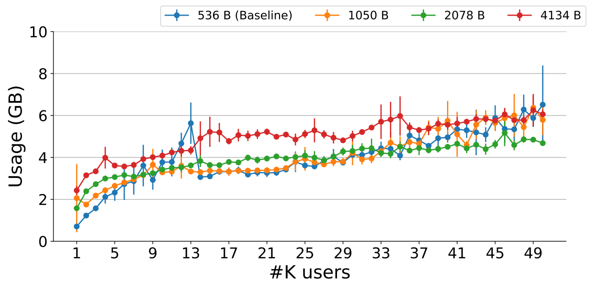


Figure 6.10: Bridge CPU usage when clients receive chaff (left) and when they receive data (right).



(a) Clients receiving chaff.



(b) Clients receiving data.

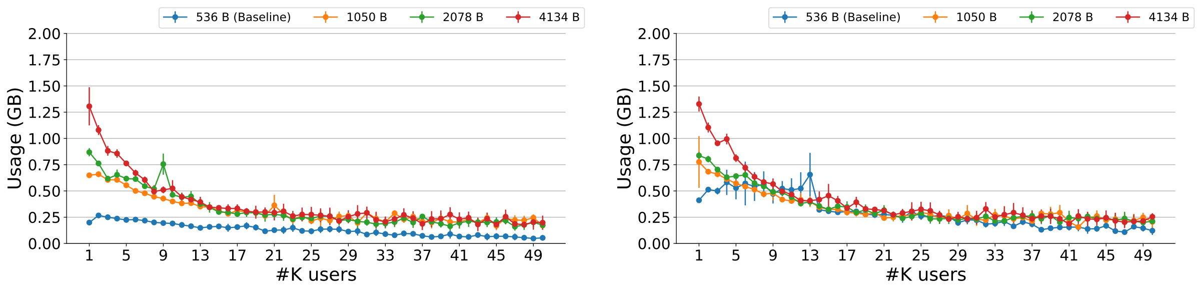
Figure 6.11: Bridge outward (towards clients and Tor Network) network usage according chunk size/TS rate.

6.4.2 Network Usage

Figure 6.11 depicts the bridge outgoing network utilization for the baseline configuration and additional chunk sizes and traffic shaping rates. The stats were collected during approximately, one minute and half, since was time taken to run each performance and latency tests individually. Usually, the higher the chunk size is, the higher the network usage will be, even though the traffic shaper function is adjusted to guarantee the same throughput across all configurations.

Figure 6.12 depicts the network usage by the client used to collect throughput and latency metrics. To gather the network statistics the command *docker stats* was used. There is a slightly increase in network usage under the baseline configuration with clients receiving data with respect to clients receiving chaff. In both scenarios the amount of traffic received by the bridge (from clients) will remain the same. However, when clients are receiving data, the bridge is additionally sending data from the iperf client to the Tor Network which will increase the network usage. In other words, when clients are receiving data, bridges are sending TorK frames to clients and, additionally, iperf's client Tor cells.

Nevertheless, it is possible to offer a reasonable bandwidth to clients without steeply increasing network usage. Nowadays, streaming HD video from the web, using 5 Mbps, which will use up to 75 MB in two minutes. Streaming a 4K video, using a 25 Mbps connection, will use up to 375 MB over a two minutes period. Therefore, with 50 connected clients, the network usage is about 125 MB (outgoing). Taking into account the both incoming and outgoing traffic, a total of 250 MB is used, slightly less than streaming a 4K video over the same period.



(a) Clients receiving chaff. (b) Clients receiving data.

Figure 6.12: Iperf client outgoing (towards bridge) network usage according chunk size/TS rate configuration.

```
int main(int argc, char* argv[]) {
    params p;
    std::string decoy = "TORK_DECOY";
    ...
    std::cerr << "[TORK] Welcome to Tork!" << std::endl;
    parse_args(argc, argv, p);
    ...
}
```

Figure 6.13: Decoy variable added to TorK.

6.5 Hardening TorK against Untrusted Bridges

Hardening TorK using SGX technologies translates in protecting the TorK memory space by adding a layer of isolation with hardware support. In this section we study the level of additional security provided by SGX enclaves and a comparison regarding performance.

6.5.1 Protection of the Bridge Runtime State

To evaluate the level of security provided by SGX enclaves, we setup a bridge environment with and without SGX, as we denote for the request of the section as the *SGX* TorK and *regular* TorK respectively. A network attacker cannot infer the number of real clients, i.e. those that are connected to the Tor Network, from those that are chaff-only based on network traces. TorK bridges handle private keys and different frame types. Usually, if the attacker aim at discovering the type of frames sent to a client, by carving into the bridge memory, he may identify real clients.

In this experiment we considered a network attacker capable of launching bridges or having high privilege access to bridges (root access). To this extent, we changed TorK source code to add a decoy variable (Figure 6.13), which in this example, may represent a private key, or the type of the last frame received by a client. Having root access, an attacker can easily identify the TorK memory regions. Figure 6.14 depicts the memory mapping associated with the TorK process, which the attacker may use to identify the stack's memory bounds. Using an ordinary debugger (Figure 6.15), it is possible to dump the stack memory region to file.

```
# cat /proc/$(pgrep tork)/maps
...
7f0c0d7a3000-7f0c0d7a4000 rw-p 00028000 08:02 38019523                   /lib/x86_64-linux-gnu/ld-2.27.so
7f0c0d7a4000-7f0c0d7a5000 rw-p 00000000 00:00 0
7ffd462ea000-7ffd4630b000 rw-p 00000000 00:00 0                          [stack]
7ffd463d2000-7ffd463d6000 r--p 00000000 00:00 0                          [vvar]
7ffd463d6000-7ffd463d8000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

Figure 6.14: TorK memory regions, whose stack lies between `0x7ffd462ea000` and `0x7ffd4630b000`.

```
# gdb --pid $(pgrep tork)
...
(gdb) dump memory /tmp/tork_mem_dumps/stack 0x7ffdeb7fd000 0x7ffdeb81e000
```

Figure 6.15: GNU Debugger (GDB) extracting a memory dump from an alive TorK bridge process.

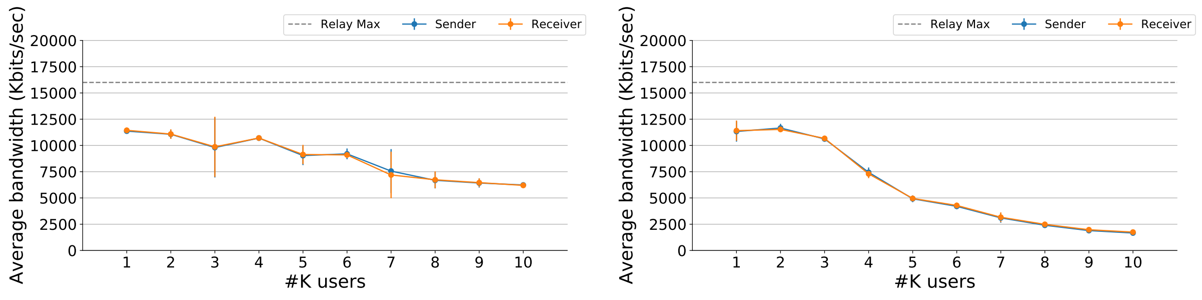
```
...
0001f4a0: 0000 0000 0000 f03f e094 3046 fd7f 0000  .......?..0F....
0001f4b0: 0093 3046 fd7f 0000 abaa aaaa aaaa ea3f  ..0F...........?
0001f4c0: d094 3046 fd7f 0000 0a00 0000 0000 0000  ..0F............
0001f4d0: 544f 524b 5f44 4543 4f59 000d 0c7f 0000  TORK_DECOY......
0001f4e0: f094 3046 fd7f 0000 0400 0000 0000 0000  ..0F............
0001f4f0: 3930 3930 007f 0000 20ac c10c 0c7f 0000  9090.... .......
0001f500: a859 c10c 0c7f 0000 13af 580d 0c7f 0000  .Y........X.....
0001f510: 0100 0000 0000 0000 18a1 0b0c 0c7f 0000  ................
0001f520: 80bd 0b0c 0c7f 0000 43b9 11a0 4656 0000  ........C...FV..
0001f530: 08b0 11a0 4656 0000 289b 3046 fd7f 0000  ....FV..(.0F....
0001f540: 789b 3046 fd7f 0000 13af 580d 0c7f 0000  x.0F......X.....
0001f550: 0100 0000 0000 0000 0000 0000 0000 0000  ................
0001f560: 7095 3046 fd7f 0000 0600 0000 0000 0000  p.0F............
0001f570: 6272 6964 6765 0000 1a28 590d 0c7f 0000  bridge...(Y.....
0001f580: 9095 3046 fd7f 0000 0000 0000 0000 0000  ..0F............
...
```

Figure 6.16: Regular stack memory dump containing the decoy variable.

```
...
0001d3a0: a1b9 7602 ac18 d071 5901 f754 bc56 c51e  ..v....qY..T.V..
0001d3b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0001d3c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0001d3d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0001d3e0: 710c 722f 6f2c d36c 51ba 6162 a6df 607c  q.r/o,.lQ.ab..‘|
0001d3f0: 37fb f2c5 f028 d64f a03c 8b0a 18f8 71d9  7....(.O.<....q.
0001d400: f143 343c 0209 4202 4019 afc0 ebb8 7e16  .C4<..B.@.....~.
0001d410: 0550 bca0 6539 7aaf 9b53 75e4 94ee 2e27  .P..e9z..Su....’
0001d420: c348 8ca0 eb0d 59c6 3006 d397 b560 eeb3  .H....Y.0....‘..
0001d430: b5a6 9077 b0ef e7c7 7fc8 e8a0 92f1 73f8  ...w..........s.
0001d440: 43ef 3320 6183 4533 6c7f ef14 b56b eae1  C.3 a.E3l....k..
0001d450: 0bf8 dd4e 9dd9 1270 a033 0324 3e4b c83b  ...N...p.3.$>K.;
0001d460: f310 69cd eb2f 5aac 4964 e6d8 cd7d 7d7f  ..i../Z.Id...}}.
0001d470: 9260 3851 ab39 59d4 0c38 2a83 33b5 97d2  .‘8Q.9Y..8*.3...
0001d480: 6666 406e 81f1 21e6 1b09 c87d 485c bd17  ff@n..!....}H\..
...
```

Figure 6.17: SGX stack memory dump.

Figure 6.16 displays the memory dump of regular TorK version, where the decoy variable is visible along with the TorK mode operation (BRIDGE). The same experiment applied to SGX version reveal a different result (Figure 6.17). Without using SGX is possible to inspect and read the content of the decoy



(a) Clients receiving chaff. (b) Clients receiving data.

Figure 6.18: Throughput of TorK-SGX baseline (536 bytes chunk) varying the number of clients.

variable from the memory dump. With use of SGX, all memory content is encrypted leaving no readable data available. We used the decoy variable as a simple proof of what an attacker can infer from reading the content of the stack. It may be trivial to identify the structure that stores the members of each k-circuit or even the reception queues. Accessing the reception queues (as described in Section 4.3.2) will lay bare which clients are receiving chaff from those who are actually sending meaningful data.

6.5.2 Performance Overheads of SGX

One of the main possible drawbacks regarding SGX is it performance impact, which is predictable to decrease due to encryption and decryption operations from and to the main memory. The baseline experiment from Section 6.2 was repeated using a capable SGX machine and the throughput results are depicted in Figure 6.18. Our nodes are again executed within Docker containers provisioned with the same virtual CPU and RAM. Bridges and clients run in separated hosts where the bridge (SGX machine) is equipped with an Intel Core i9-9900K CPU. However due to hardware limitations, the SGX experiment could only be tested with 10 clients. Thus, the TS rate window used was 50 µs to 500 µs to accommodate the same theoretical throughput.

Notwithstanding the memory protection offer by SGX we study the possibility to perform remote attestation of both clients and bridges. It is currently, an important weakness point liable to be explored by attackers. Unfortunately, SCONE prototype does not support remote attestation [56]. As a result, the full implementation of a TorK prototype featuring remote attestation is left for future work.

Summary

This chapter presented our experimental evaluation of TorK. It showed that TorK is able to generate a indistinguishable channel. TorK is able to provide a reasonable throughput and latency needed for typical network activities (e.g. browsing). TorK has shown to be resilient against correlation attacks. The presence of a powerful active attacker performing packet drops or delays are not sufficient to identify clients. However, depending on the configuration parameters, network usage or users' performance can be affected. The next chapter concludes this documents by reviewing the key points of this thesis and introduce the future work.

Chapter 7

Conclusions

The current Tor specification is vulnerable to traffic correlation attacks. An adversary that has access to multiple hops in a circuit may match a request back to the sender, defeating the very purpose of Tor as an anonymity system. To date, the most successful correlation attacks explore network flows' inter-packet arrival times and apply machine learning-based approaches that achieve promising results despite the unpredictable Tor network environment (e.g. network delays). Recent efforts proposed in the literature make it more difficult for adversaries to succeed in traffic correlation. Such approaches do not attempt to change the characteristics of individual Tor flows, but employ a number of strategies that decrease users' probability to select malicious relays and that avoid routing traffic through malicious autonomous systems altogether.

We proposed a *k-anonymity* approach to Tor in order to make it robust against known traffic correlation attacks. To the best of our knowledge, we are the first to propose applying the k-anonymity concept to a low-latency circuit based anonymous system to prevent an adversary from identifying a sender in a group of k other users. By providing an indistinguishable channel along with a set of synchronization protocols and measures to withstand against active attacks, TorK offers additional protection against correlation attacks. Using the current Tor pluggable transport API, TorK is fully compatible with Tor existing infrastructure. The main challenges tackled when building TorK were tied to i) the efficient implementation of TorK while adhering to the Tor API and ii) the implementation of TorK reactive measures (e.g. bandwidth control) while maintaining reasonable throughput, latency, and assuring indistinguishability and unlinkability of the network channels while using reasonable network usage and CPU resources.

7.1 Achievements

The major achievement of the present work is the design and implementation of TorK, a Tor pluggable transport and controller which offers additional protection against traffic correlation attacks by providing k-anonymity. By forming a synchronized group of K users, an adversary can no longer match the source of a Tor circuit with its destination. Other main accomplishments of this thesis encompass: i) guaranteeing the indistinguishablility of TorK flows; ii) achieving reasonable throughput and latency on TorK flows;

iii) providing resilience against passive and active attacks, iv) a meticulous study of the implications of TorK configuration parameters on throughput, latency, the use of CPU and network resources, and; v) exploring the execution of TorK under a secure environment, provided by hardware extensions, so as to increase the security of the system against malicious bridges and clients.

7.2 Future Work

Our work on TorK allows us to identify multiple interesting directions for future work. First, the implementation of TorK should be enhanced with the support for remote attestation, which in turn would help bridges and clients to attest that communicating parties are running unmodified TorK releases. Second, TorK's performance-sensitive operations like bridges' load balancing behavior and k-circuit rearrangement requires further assessment when facing a large pool of users. Specifically, better load balancing capabilities may help resource-constrained bridges to provide a reasonable quality of service. In its turn, K-circuits rearrangements – presented as a constrained optimization problem – alone are a solid research topic demanding a exhaustive analysis and a comprehensive algorithm to find the best balance between waiting for new users without disrupting current ones vs. redistributing new users among existing k-circuits to prevent starvation. Third, future work should also focus on other strategies to create and manage indistinguishable channels, for instance, by taking advantage of spared network approaches which may be meaningful for users with network restrictions (e.g. mobile networking). A last topic for envisioned future work tackles the incentives for participation in the TorK ecosystem. To this end, it may be important to design and implement an effective reward system.

Bibliography

[1] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, Aug 2004.

[2] Tor Project. Tor faq. `https://2019.www.torproject.org/about/overview.html.en`, 2019. Accessed: 2020-01-05.

[3] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *ACM SIGSAC Conference on Computer and Communications Security*, Nov 2013.

[4] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira. Measuring and mitigating as-level adversaries against tor. In *Network and Distributed System Security Symposium (NDSS)*, Feb 2016.

[5] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information. In *Principles of Database Systems Symposium (PODS)*, Jun 1998.

[6] Tor Project. Tor pluggable transports. `https://2019.www.torproject.org/docs/pluggable-transports`, 2019. Accessed: 2020-01-05.

[7] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Network and Distributed System Security Symposium (NDSS)*, Feb 2009.

[8] Tor Project. Tor directory specification. `https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt`, 2020. Accessed: 2020-01-05.

[9] S. Matic, C. Troncoso, and J. Caballero. Dissecting tor bridges: A security evaluation of their private and public infrastructures. In *Network and Distributed System Security Symposium (NDSS)*, Feb 2017.

[10] I. E. T. Force. Rfc 6066: Transport layer security (tls) extensions: Extension definitions. `https://tools.ietf.org/html/rfc6066`, Jan. 2011. Accessed: 2020-01-05.

[11] Tor Project. A child garden of pluggable transports. `https://trac.torproject.org/projects/tor/wiki/doc/AChildsGardenOfPluggableTransports`, May 2016. Accessed: 2020-01-05.

[12] Tor Project. meek. https://trac.torproject.org/projects/tor/wiki/doc/meek, May 2019. Accessed: 2020-01-05.

[13] Y. Angel. obfs4 (the obfourscator). https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt, Jan. 2019. Accessed: 2020-01-05.

[14] Tor Project. obfs2 (the twobfuscator). https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs2/obfs2-protocol-spec.txt, Jan. 2015. Accessed: 2020-01-05.

[15] S. Chakravarty, A. Stavrou, and A. D. Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *European Symposium on Research in Computer Security*, Sep 2010.

[16] R. Pries, W. Yu, X. Fu, and W. Zhao. A new replay attack against anonymous communication networks. In *IEEE International Conference on Communications*, May 2008. doi: 10.1109/ICC.2008.305.

[17] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *IEEE Symposium on Security and Privacy*, May 2005.

[18] B. Evers, J. Hols, E. Kula, J. Schouten, M. den Toom, R. van der Laan, and J. Pouwelse. Thirteen years of tor attacks. *Computer Science, Delft University of Technology*, Nov. 2019.

[19] P. Winter, R. Ensafi, K. Loesing, and N. Feamster. Identifying and characterizing sybils in the tor network. In *USENIX Security Symposium*, Aug 2016.

[20] J. R. Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, Mar 2002.

[21] K. Bauer, D. McCoy, D. C. Grunwald, and T. Kohno. Low-resource routing attacks against anonymous systems. In *ACM Workshop on Privacy in the Electronic Society*, Oct 2007.

[22] D. R. Figueiredo, P. Nain, and D. Towsley. On the analysis of the predecessor attack on anonymity systems. *Computer Science Technical Report*, Aug 2004.

[23] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending anonymous communications against passive logging attacks. In *IEEE Symposium on Security and Privacy*, May 2003.

[24] R. Dingledine and G. Kadianakis. One fast guard for life (or 9 months). In *Privacy Enhancing Technologies*, Jul 2014.

[25] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. RAPTOR: Routing attacks on privacy in tor. In *USENIX Security Symposium*, Washington, D.C., Aug 2015. URL https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sun.

[26] V. Shmatikov and M.-H. Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *European Symposium on Research in Computer Security*, Jan 2006.

[27] M. Nasr, A. Bahramali, and A. Houmansadr. Deepcorr. *ACM SIGSAC Conference on Computer and Communications Security*, Oct 2018. doi: 10.1145/3243734.3243824. URL `http://dx.doi.org/10.1145/3243734.3243824`.

[28] M. Akhoondi, C. Yu, and H. V. Madhyastha. Lastor: A low-latency as-aware tor client. In *IEEE Symposium on Security and Privacy*, May 2012.

[29] M. Edman and P. Syverson. As-awareness in tor path selection. In *ACM SIGSAC Conference on Computer and Communications Security*, Nov 2009.

[30] Tor Project. Tor faq. `https://2019.www.torproject.org/docs/faq.html.en`, 2019. Accessed: 2020-01-05.

[31] Tor Project. Reporting bad relays. `https://trac.torproject.org/projects/tor/wiki/doc/ReportingBadRelays`, Sep 2018. Accessed: 2020-01-05.

[32] Tor Project. How to report bad relays. `https://blog.torproject.org/how-report-bad-relays`, Jul 2014. Accessed: 2020-01-05.

[33] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. Weippl. Spoiled onions: Exposing malicious tor exit relays. In *Privacy Enhancing Technologies*, Jul 2014.

[34] S. Chakravarty, G. Portokalidis, M. Polychronakis, and A. D. Keromytis. Detecting traffic snooping in tor using decoys. In *International Workshop on Recent Advances in Intrusion Detection*, Sep 2011.

[35] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg. Changing of the guards: A framework for understanding and improving entry guard selection in tor. In *ACM Workshop on Privacy in the Electronic Society*, Oct 2012.

[36] J. Juen, A. Johnson, A. Das, N. Borisov, and M. Caesar. Defending tor from network adversaries: A case study of network path prediction. *Privacy Enhancing Technologies*, Jun 2015.

[37] C. Wacek, H. Tan, K. S. Bauer, and M. Sherr. An empirical evaluation of relay selection in tor. In *Network and Distributed System Security Symposium (NDSS)*, Feb 2013.

[38] Z. Li, S. Herwig, and D. Levin. Detor: Provably avoiding geographic regions in tor. In *USENIX Security Symposium*, Aug 2017.

[39] D. Levin, Y. Lee, L. Valenta, Z. Li, V. Lai, C. Lumezanu, N. Spring, and B. Bhattacharjee. Alibi routing. In *ACM Special Interest Group on Data Communication*, Aug 2015.

[40] Z. Weinberg, S. Cho, N. Christin, V. Sekar, and P. Gill. How to catch when proxies lie: Verifying the physical locations of network proxies with active geolocation. In *Internet Measurement Conference*, New York, NY, Oct 2018. Association for Computing Machinery.

[41] K. Kohls, K. Jansen, D. Rupprecht, T. Holz, and C. Pöpper. On the challenges of geographical avoidance for tor. In *Network and Distributed System Security Symposium (NDSS)*, Feb 2019.

[42] Tor Project. What are entry guards? https://2019.www.torproject.org/docs/faq.html.en#EntryGuards, 2019. Accessed: 2020-01-05.

[43] K. Bock, G. Hughey, X. Qiang, and D. Levin. Geneva: Evolving censorship evasion strategies. In *ACM SIGSAC Conference on Computer and Communications Security*, Nov 2019.

[44] B. Schneier. How the nsa attacks tor/firefox users with quantum and foxacid. https://www.schneier.com/blog/archives/2013/10/how_the_nsa_att.html, Oct 2013. Accessed: 2020-01-05.

[45] B. Kenig and T. Tassa. A practical approximation algorithm for optimal k-anonymity. *Data Mining and Knowledge Discovery*, Jul 2012. doi: 10.1007/s10618-011-0235-9. URL http://dx.doi.org/10.1007/s10618-011-0235-9.

[46] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, May 2008.

[47] C. Dwork. Differential privacy. In *33rd International Conference on Automata, Languages and Programming - Volume Part II*, Berlin, (DE), July 2006. ISBN 3-540-35907-9, 978-3-540-35907-4. doi: 10.1007/11787006_1. URL http://dx.doi.org/10.1007/11787006_1.

[48] R. Shokri, C. Troncoso, C. Diaz, J. Freudiger, and J.-P. Hubaux. Unraveling an old cloak: K-anonymity for location privacy. In *ACM Workshop on Privacy in the Electronic Society*, Oct 2010. ISBN 978-1-4503-0096-4. doi: 10.1145/1866919.1866936. URL http://doi.acm.org/10.1145/1866919.1866936.

[49] A. Campan and T. M. Truta. Data and structural k-anonymity in social networks. In *Privacy, Security, and Trust in KDD: Second ACM SIGKDD International Workshop*, Aug 2009. ISBN 978-3-642-01717-9.

[50] A. Gupta and N. Shukla. Privacy preservation in big data using k-anonymity algorithm with privacy key. *International Journal of Computer Applications*, Nov 2016.

[51] D. Barradas, N. Santos, and L. Rodrigues. Effective detection of multimedia protocol tunneling using machine learning. In *USENIX Security Symposium*, Aug 2018. ISBN 978-1-931971-46-1. URL http://dl.acm.org/citation.cfm?id=3277203.3277217.

[52] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The loopix anonymity system. In *USENIX Security Symposium*, Aug 2017. ISBN 978-1-931971-40-9. URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska.

[53] S. Kim, J. Han, J. Ha, T. Kim, and D. Han. Enhancing security and privacy of tor's ecosystem by using trusted execution environments. In *USENIX on Networked Systems Design and Implementation*, Boston, MA, Mar 2017.

[54] Tor Project. Tor control protocol. https://gitweb.torproject.org/torspec.git/tree/control-spec.txt, Jul. 2020. Accessed: 2020-07-24.

[55] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. Rfc 1928: Socks protocol version 5. `https://tools.ietf.org/html/rfc1928`, Mar. 1996. Accessed: 2020-01-05.

[56] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell, et al. {SCONE}: Secure linux containers with intel {SGX}. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 689–703, 2016.

[57] J. Hayes and G. Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *Proceedings of the 25th USENIX Security Symposium*, pages 1187–1203, Austin, TX, USA, 2016.

[58] Tor Project. Tor metrics - relay search. `https://metrics.torproject.org/rs.html#toprelays`, 2018. Accessed: 2020-07-27.

[59] N. Mathewson. Allow 4-byte circuit ids in a new link protocol, Nov 2012. URL `https://gitweb.torproject.org/torspec.git/tree/proposals/214-longer-circids.txt`.