QATAR UNIVERSITY

COLLEGE OF ENGINEERING

ENHANCING THE PERFORMANCE AND SECURITY OF ANONYMOUS COMMUNICATION NETWORKS

BY

LAMIAA MOHAMED BASYONI

A Dissertation Submitted to

the College of Engineering

in Partial Fulfillment of the Requirements for the Degree of

Doctorate of Philosophy in Computer Science

January 2022

COMMITTEE PAGE

The members of the Committee approve the Dissertation of
Lamiaa Mohamed Basyoni defended on 30/11/2021.

Prof. Mohsen Guizani
Dissertation Supervisor

Dr. Aiman Erbad
Dissertation Co-Supervisor

Dr. Amr Mohamed
Committee Member

Dr. Khaled Khan
Committee Member

Dr. Elias Yaacoub
Committee Member

Approved:

Khalid Kamal Naji, Dean, College of Engineering

ABSTRACT

Basyoni, Lamiaa, Mohamed, PhD: January: 2022, Doctorate of Philosophy in Computer Science

Title: Enhancing the Performance and Security of Anonymous Communication Networks.

Supervisor of Thesis: Mohsen, Guizani. Aiman, Erbad.

With the increasing importance of the Internet in our daily lives, the private information of millions of users is prone to more security risks. Users' data are collected either for commercial purposes and sold by service providers to marketeers or political purposes and used to track people by governments, or even for personal purposes by hackers. Protecting online users' privacy has become a more pressing matter over the years. To this end, anonymous communication networks were developed to serve this purpose.

Tor's anonymity network is one of the most widely used anonymity networks online; it consists of thousands of routers run by volunteers. Tor preserves the anonymity of its users by relaying the traffic through a number of routers (called onion routers) forming a circuit. Tor was mainly developed as a low-latency network to support interactive applications such as web browsing and messaging applications. However, due to some deficiencies in the original design of Tor's network, the performance is affected to the point that interactive applications cannot tolerate it. In this thesis, we attempt to address a number of the performance-limiting issues in Tor network's design.

Several researches proposed changes in the transport design to eliminate the effect of

these problems and improve the performance of Tor's network. In our work, we propose "QuicTor," an improvement to the transport layer of Tor's network by using Google's protocol "QUIC" instead of TCP. QUIC was mainly developed to eliminate TCP's latency introduced from the handshaking delays and the head-of-line blocking problem. We provide an empirical evaluation of our proposed design and compare it to two other proposed designs, IMUX and PCTCP. We show that QuicTor significantly enhances the performance of Tor's network.

Tor was mainly developed as a low-latency network to support interactive web browsing and messaging applications. However, a considerable percentage of Tor traffic is consumed by bandwidth acquisitive applications such as BitTorrent. This results in an unfair allocation of the available bandwidth and significant degradation in the Quality-of-service (QoS) delivered to users. In this thesis, we present a QoS-aware deep reinforcement learning approach for Tor's circuit scheduling (QDRL). We propose a design that coalesces the two scheduling levels originally presented in Tor and addresses it as a single resource allocation problem. We use the QoS requirements of different applications to set the weight of active circuits passing through a relay. Furthermore, we propose a set of approaches to achieve the optimal trade-off between system fairness and efficiency. We designed and implemented a reinforcement-learning-based scheduling approach (*TRLS*), a convex-optimization-based scheduling approach (*CVX-OPT*), and an average-rate-based proportionally fair heuristic (*AR-PF*). We also compared the proposed approaches with basic heuristics and with the implemented scheduler in Tor. We show that our reinforcement-learning-based approach (TRLS) achieved the highest

QoS-aware fairness level with a resilient performance to the changes in an environment with a dynamic nature, such as the Tor network.

DEDICATION

To my family, my amazing husband, and my beloved Ali and Farida, for their love, support, and faith in me.

To my grandmother.

ACKNOWLEDGMENTS

I would like to sincerely thank my Thesis supervisors, Prof. Mohsen Guizani and Dr. Aiman Erbad, for encouraging me to deliver my best and believing in me. I would also like to thank Prof. Amr Mohamed for his valuable feedback throughout the Thesis. This work would not have been possible if it were not for the inspiration of Dr. Mashael AlSabah. I want to thank her so much.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

$\mathcal{A}$	Action space
γ	Discount factor
$\mathcal{P}$	Probability density function
π	Policy
$\mathcal{R}$	Stochastic reward function
$\mathcal{S}$	State space
$\mathcal{T}$	State transition function
A	Action random variable
a	Sample action value
$A(s, a)$	The advantage function for a policy π
$Q^{\pi}(s, a'; \theta)$	The approximation of the state-action value function for a policy π using parameter vector θ
$Q^{\pi}(s, a)$	The state-action value function for a policy π
$Q^{\pi^*}(s, a), Q^*(s, a)$	The approximation of the optimal state-action value
$Q^{\pi^*}(s, a; \theta), Q^*(s, a; \theta)$	The approximation of the optimal state-action value function using parameter vector θ
R	Cumulative reward
r	Sample reward value
S	State random variable
s	Sample state value
$V(s)$	The state value function for a policy π

CHAPTER 1: INTRODUCTION

Motivation

The Internet has emerged as the fastest way to access information and build services in the past decades. However, this revolutionary information age comes with its own set of challenges. The privacy of Internet users is at increasing risk with the advances in censorship and surveillance techniques. Internet Service Providers (ISPs) and large tech companies track users' online behavior, activities, and even personal information to strategize their marketing plans at the expense of users' privacy. Moreover, new attacks can use botnets to steal log-in credentials, launch Denial of Service (DoS) attacks, and send phishing spam mail [1]. Even governments can misuse the information about the user identities and online activities; people were arrested for expressing their opinions in tens of countries around the world [2]. Governments of many countries are either controlling the ISPs or forcing them to block access to certain websites based on the government policy [3]. Due to its importance, the United Nations Human Rights Council declared the Internet to be key means through which individuals may exercise their freedom of expression [4]. This declaration enlisted online privacy as a human right that should be protected.

Privacy-preserving technologies were developed in response to the increasing need to preserve and protect online users' privacy. A famous example of the technologies developed to protect the confidentiality and privacy of the users' information by encrypting the communication between a client and a server is Virtual Private Networks (VPNs). However, in this case, the VPN provider controls all the traffic and has access to it. Moreover, some services are blocking VPNs and cannot be reached through them [5].

Anonymity networks were then introduced as a more efficient technique to preserve the users' online anonymity. Anonymity networks hide the links between the online user's IP address and his online activities. Some of the earliest developed anonymity networks were built on the concept introduced by Chaum [6], the main goal was to maintain the anonymity of users on the Internet, which came at the expense of introducing long and intolerable latency [7]. This type of anonymity network, called *high-latency* networks, was not suitable for interactive applications such as web browsing. A different type of anonymity network that was mainly designed to support such interactive applications was the *low-latency* anonymity network.

Tor [8] anonymity network is a low-latency anonymity network that has gained an excellent reputation over the past years and is being adopted by millions of users. Tor was first introduced in 2003 and had been growing in terms of the number of running routers and supported users. As per the statistics from Tor's live network[9], in 2019, the number of directly connected users, not including those connecting through bridges, exceeded 3 million users, the number of operating relays reached 6500 relays and more than 1000 bridges, and more than 60 thousands unique .onion addresses were available for hidden services.

Tor anonymity network is designed based on the concept of *Onion Routing* [10], [11], to hide the link between the source and destination of TCP traffic. The significant increase in the usage of the Tor anonymity network brought to light the fact that, while Tor is powerful in hiding user's identities and protecting their privacy, it suffers from performance issues introducing a delay that is unacceptable, especially for new interactive applications [12], [13].

Achieving an acceptable performance of Tor's anonymity network does does not only

affect the user's experience, but also the security and anonymity of the network in many ways. Usability is known to be an essential factor of security [14]. The growth of Tor's network resource, the volunteered relays, is affected by how well it is utilized and how the load-balancing is handled. The growth of Tor's network resources can be directly related to the level of anonymity provided by the network. Several studies addressed the performance problems in Tor [15], [16]. The main goal of these studies was to identify the sources of delay in the network. A clear understanding of the delay causes would lead to a more informed design of Tor's network and help enhance the overall performance.

Problem Statement

One of the design weaknesses in Tor that negatively affects the performance of the network is its *Transport Design*. Tor maintains a single TCP connection between each two communicating *Onion Routers (ORs)*. Tor then multiplexes traffic from multiple sources over the same *OR-to-OR* connection. A reliable in-order transport protocol like TCP suffers from problems such as *head-of-line-blocking* that could halt the connection, especially in a lossy connection. Because of Tor's transport design, this type of problems would drastically affect the network's overall performance. Motivated by this knowledge, the Tor community started considering using datagram protocols as the base for the transport layer [17], [18]. Google's new datagram-based protocol *QUIC* has been perceived as a sufficient alternative for TCP to avoid the performance limitation [19]. Another problem in Tor's design that was shown to be affecting the network performance is the scheduling mechanisms. Although Tor was mainly developed as a low-latency network to support interactive applications such as web browsing and messaging ap-

plications, a considerable percentage of Tor traffic is consumed by bandwidth-intensive applications such as *BitTorrent* [20], [21], which leads to unfair allocation of the available bandwidth and degrades the *Quality-of-service (QoS)* delivered to users. To understand the effects of the file-sharing applications on the network performance, the study in [22] showed that while file-sharing applications represent 3% to 4% of the number of connections, they consume up to 40% of the available bandwidth. The issue of utilizing Tor's volunteered resources has increased further with the use of Tor by *botnets*[23]. *QoS-aware* fair scheduling to Tor's circuits will help utilize the network resources to improve the overall performance of the network.

The challenge of scheduling and resource allocation in the case of Tor lies in the fact that it is done at two levels. At the connections level, the original design of Tor writes from one connection at a time. It writes as much data as possible, causing a problem known as buffer-bloating, where the send buffer gets bloated with traffic to respect the TCP semantics of reliable in-order congestion-controlled delivery. TCP implementation in Linux uses auto-tuning to increase the capacity of each socket buffer when needed. Tor relays maintain hundreds of TCP sockets to all connected relays. When TCP runs auto-tuning and increases the buffer capacity for each socket to accommodate the data written from the Tor application, the kernel will not be able to send it all to the network, increasing the system's delay. With so many active sockets, the kernel queuing delay will be unbearable. This behavior was later modified by imposing writing limits. However, it did not consider the QoS requirements of the circuit level. The circuits mapped to the chosen writable connection are then being managed using the circuit level scheduler. While attempting to be fair at each level, this scheme does not accomplish overall fairness to all circuits running on Tor's relay. Moreover, in an overlay network like Tor,

connections are not mapped to separate physical links. They are to be competing with traffic from other links on the configured bandwidth. Artificial intelligence techniques such as *reinforcement learning (RL)* has been used for resource allocation in the last few years and was proven to be efficient [24].

Path Selection Algorithm in Tor refers to the process by which Tor selects the onion routers to be used in building virtual circuits. Tor's path selection strategy is selecting the relays for circuit construction unawares of the underlying Internet routing. This design leads to two main problems; degraded performance and compromised anonymity. A path selection algorithm should find the balance between performance and security.

Thesis Objectives and Contribution

Our work is motivated by the increasing need to protect users' privacy on the Internet. Tor is the defacto anonymity network. It suffers from the identified performance limitations affecting the privacy and anonymity of the network; The main objective of this research is to enhance the Tor network's real-time performance to promote its adoption by more online users. To fulfill this objective, we contribute solutions to three of the design weaknesses in the Tor network.

1. Addressing Tor transport design performance limiting issues. To that end, we:

    - Propose QuicTor as a suitable design of a transport-level datagram protocol for Tor using Google's QUIC protocol.
    - Build a realistic test-bed that supports the use of UDP-based protocols (e.g., QUIC) and calibrate our environment using the performance of Tor's live network.

- Evaluate the performance gain of QuicTor over vanilla Tor for different types of applications (web browsing, bulk downloads, and video streaming).
- Present a comprehensive study of the security of QuicTor by analyzing different categories of attacks on the Tor network. We implemented diverse types of attacks and assessed their impact on QuicTor in comparison to vanilla Tor.

2. Addressing Tor's unfair circuit scheduling. To that end, we:

   - Design a *QoS-aware* scheduling scheme for Tor network streams aiming to achieve an optimal trade-off between fairness and efficiency according to traffic type.
   - Formulate Tor's circuit scheduling as a convex-optimization problem whose solution achieved optimal trade-off between fairness and delivered QoS.
   - Propose a new heuristic based on the weighted moving average (WMV) for proportional fair scheduling of Tor circuits.
   - Design a Deep Reinforcement Learning *agent* to achieve fairness-QoS trade-off in real-time and based on dynamic and flexible user requirements.
   - Perform rigorous evaluation of the proposed algorithms against the state-of-the-art techniques.

Thesis Overview

This chapter motivated the Thesis topic and summarised its objectives and contributions. The remainder of the dissertation is organized as follows: In Chapter 2, we introduce the main concepts, terminologies, and frameworks. A survey of related

work and contrasting them to our work is presented in chapter 3. In chapter 4, we present the design and implementation of *QuicTor* our datagram-based transport layer for Tor using QUIC protocol and empirically evaluate the performance and security of QuicTor. Chapter 5 presents the formulation and evaluation of the proposed solutions for QoS-aware circuit scheduling in Tor; we evaluate the proposed approaches against state-of-the-art methods. Chapter 6 presents our plans for future research directions. Finally, we conclude the work, summarize the main results, state limitations and potential improvements in Chapter 7.

CHAPTER 2: BACKGROUND

Since the introduction of Chaum's mix-net, developing anonymous communication networks has advanced in two directions, based on the type of applications supported; high-latency and low-latency anonymous networks. In this chapter, we briefly describe the basics of anonymous networks. Then we present a detailed background of Tor, the most popular low-latency anonymity network.

Anonymous Communication Networks

The objective of anonymous communication networks is to preserve the confidentiality of both ends of the traffic. Accordingly, anonymous communication networks were designed to distribute the communication between the users and their destination over multiple hops along the way. Messages passed on from one hop to another are usually fixed size and cryptographically handled at each hop.

The main difference between high-latency and low-latency anonymity networks is the added delay in high-latency networks; the added delay is caused by intentionally storing the data and the cryptographic alteration done to mitigate attacks. This mitigation mechanism is because the threat model of high-latency networks assumes a global adversary who can actively add or modify the traffic. On the other hand, low-latency anonymity networks define the adversary in their threat model as local and can only control part of the network; yet the adversary can also add and modify the traffic. A comparison of high-latency and low-latency anonymous communication networks is shown in table 2.1

Table 2.1: Comparison of high-latency and low-latency anonymous communication networks

	Low-latency	High-latency
Threat Model	Weak Local Adversary (active/passive)	Well-funded Global Adversary (active/passive)
Networks Examples	Crowds[25], Freedom[26], Tor[27]	Bable[28], Mixminion[29]
Applications Examples	Web Browsing, Instant Messaging, Secure Shell	E-mail, E-voting

Tor: The Onion Router

Tor is an overlay low-latency anonymity network that consists of thousands of interconnected *Onion Routers (ORs)* over which the traffic from the users to their destinations is being distributed. Each OR builds a descriptor summary containing its IP address, offered bandwidth, and encryption keys. All descriptors are sent to the *authority directory* routers. On the client-side, the process running is called *Onion Proxy (OP)*. An OP learns the required details to establish a connection to the Tor network by contacting an authority directory to obtain the *router descriptors*. Communications from the client OP to the destination server are done over virtual paths called *circuits*. A typical circuit consists of three ORs (hops); the first OR is called the *entry guard*, next is the *middle relay*, and the last hop before reaching the destination is called the *exit relay*.

Tor's communications use a fixed size *cells*, the cell size is 512 bytes. The idea of using a fixed cell size is to add some resistance to some types of attacks, such as traffic analysis. However, it was found to be inefficient and results in a distinctive distribution of the packet size in a specific stream [27]. Hence, control and padding cells are used

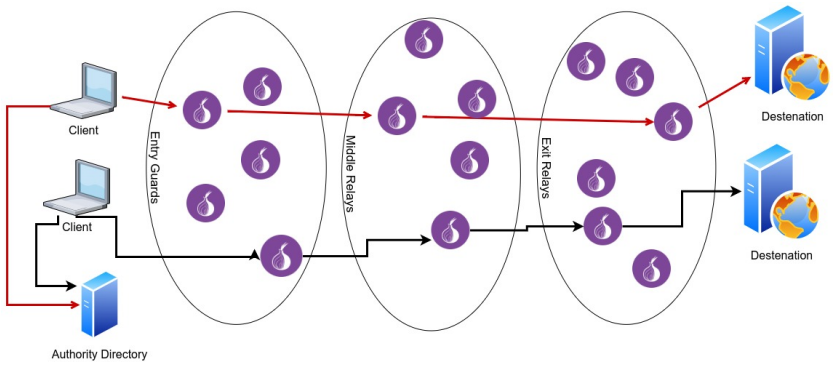


Figure 2.1: Tor's Network

with variable lengths to limit the information leak. The typical structure of the cell is shown in figure 2.2. The circuit ID and command fields are not encrypted; hence it can be processed by all ORs along the circuit to allocate the cell to the corresponding circuit queue. The remaining fields of the cell are encrypted, as mentioned earlier, and can only be processed at the exit OR. The entire cell is then encapsulated in the payload of the transport packet to be sent over the Internet. [27].

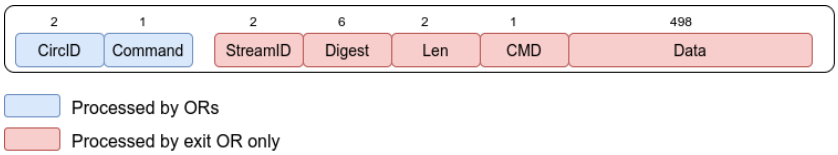


Figure 2.2: Tor's cell

Circuit Construction

To boost the performance, the OP creates several spare circuits for the user's applications. Once any of the users' applications attempt to establish a new TCP stream, the OP tries first to attach it to an existing circuit; if there are no available circuits, the OP constructs a new circuit using the following procedure.

The OP establishes connections to the first *(entry)* OR requesting the build of a circuit to the destination. The entry OR then *extends* the circuit to the next hop, until it reaches the *exit* OR. Figure 2.3 illustrates how Tor builds its circuits until the client is connected to the destination. The onion proxy accepts TCP requests (TCP *streams*) and then multiplexes them over the created circuits. The traffic from the client is encrypted with three keys, one for each hop on the circuit. At the exit OR, the destination address is revealed so the exit OR can complete the connection.

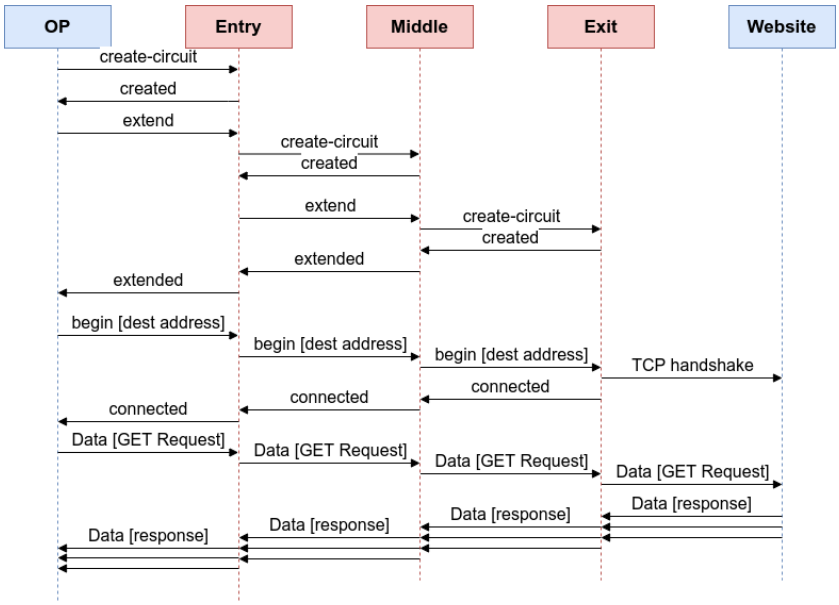


Figure 2.3: Tor's circuit establishment

Circuit Multiplexing

In Tor, OPs and ORs communicate with each other over TCP connections. Every pair of communicating ORs maintain a single TCP connection and use it for all of their communications. The original *Onion Routing* design uses one circuit per TCP stream. However, due to the latency cost of this approach, Tor is multiplexing multiple TCP streams over the same circuit.

Figure 2.4 explains the circuit multiplexing design in Tor. We can see that *client 1* and *client 2* are requesting connection to two different destinations, the OP service on each client machine creates a circuit to the desired destinations (*circ1 and circ2*). In the studied case, both circuits share the middle and exit relays. Recalling that all communication between two onion routers is done over a single connection, packets from circ1 and circ2 will share the middle-exit connection. Thus, packets from both circuits will be copied to the same output buffer on the middle relay and read from the same input buffer on the exit relay. The design might not be problematic for a small number of circuits. However, with the low relay-to-client ratio in the Tor network, thousands of circuits are multiplexed over one TCP connection, which affects the overall performance.

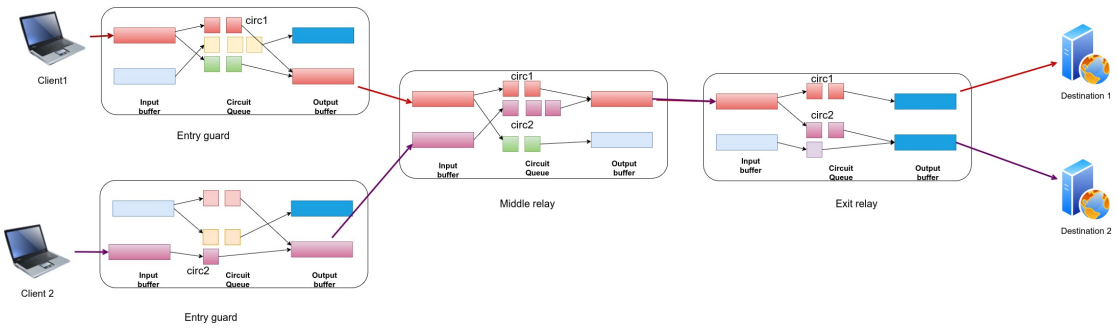


Figure 2.4: Tor's cross-circuit interference

Tor's Queuing Architecture

In the context of Tor, *connections* are the TCP/TLS connection between communicating routers, while *circuits* are the logical end-to-end connections between the client and the destination. A circuit typically travels through multiple connections along its path. The nested hierarchy of the Tor scheduler is shown in figure 2.5. A TCP con-

nection writes to the kernel buffer that is handled according to the operating system implementation. Data is then copied to the application layer, where it is stored in an application-layer buffer called the *input buffer*. Tor the direct every packet based on the circID in its header to the appropriate *circuit queue*. Circuit queues are also application-layer buffers used to store packets mapped to each circuit separately.

Outgoing packets are copied from the circuit queues to Tor's *output buffers*. Tor then writes data from the output buffer to the socket kernel buffer to be flushed to the TCP connection.

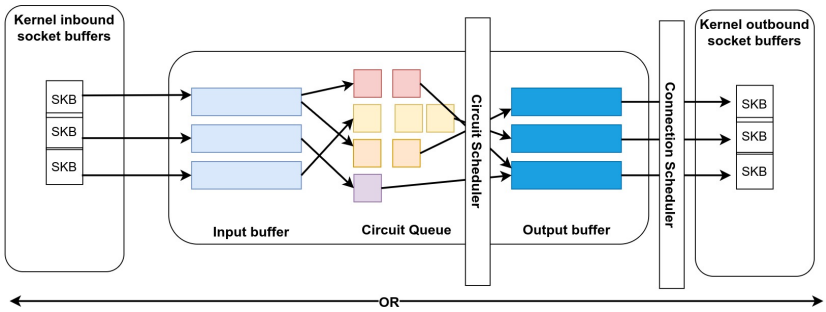


Figure 2.5: The Nested Hierarchy of Tor Scheduling Scheme

Path Selection Design

Tor chooses nodes for its circuits based on their advertised bandwidth. Relays with higher bandwidth are more likely to get more circuits. However, relay selection criteria vary according to the relay position along the circuit path. As discussed earlier, a typical circuit consists of three relays; *entry guard, middle relay, and exit relay*. Each Tor's client chooses a subset of the available relays to be its *entry guards* set and repeatedly chooses one of these relays, the entry guard, whenever this client constructs a circuit. *Exit relays* can be only chosen from the set of relays accepting the *exit policy* of Tor. Any other relay that is not an entry or an exit can be chosen to be the *middle relay* on the

circuit path. One important notice here is that communication between two consecutive nodes on the circuit path is hardly ever direct. The Internet consists of thousands of connected *autonomous systems (ASes)*. Data send from one node to another usually travel across a number of these Ases.

Tor's Threat Model

In low-latency anonymity networks such as Tor, an adversary generally aims to confirm the communication's source and destination. The current design of Tor's network assumes the absence of a global adversary that can monitor both ends of the communication, entry, and exit guards and does not provide anonymity against this type of adversaries. Instead, Tor's threat model assumes an adversary that can observe only a fraction of the communication and can control only a fraction of Tor nodes, either by running his ORs or compromising an already running ORs [30]. Based on the proposed threat model, attacks can be categorized according to their model's practicality, based on the assumptions made and the required resources to enable the attack. In the analysis of the security of the onion router provided by Syverson, *et al.* [31], the probability of facing an adversary compromising either the first node or the last node on the route is the same and is equal to (the number of compromised nodes (c)) / (the total active nodes in the system (n)). However, an adversary compromising the first *and* the last nodes on the route might exist with probability = c^2/n^2. These probabilities are valid for a particular time duration; routes are dynamic in Tor, which means that after a specific amount of time, if the path is not used to exchange traffic, it will be automatically torn down, and another path will be established for future activities.

Tor's Defenses

Tor encrypts the client's traffic using multiple keys and implements perfect forward secrecy. Hence, for an attacker to compromise the encryption keys, he has to learn the OR's TLS session key and the circuit session key, and due to the periodic rotation of the circuit key, the window available to launch such attacks is minimal. Tor also forces circuit lifetime limit, after which the ORs will erase all information that an adversary who compromises an OR on the circuit needs to carry the attacks further and compromise more nodes. In addition, Tor provides solid defenses against tagging attacks and replay attacks.

On the other hand, traffic confirmation attacks are generally out of the scope of Tor's design. Therefore, Tor provides minimal protection against adversaries aiming to correlate end-to-end timing and packet size. Website fingerprinting is another potentially effective attack on Tor's network, in which an attacker collects *"fingerprint"* of highly targeted websites containing the access patterns and tries to map observed traffic to these fingerprints [27]. Tor developed a defense against fingerprinting attacks by enabling HTTP pipelining. However, later research proved that the proposed defense was not effective in preventing fingerprinting attacks [32][33].

QUIC Transport Protocol

For decades, TCP has been the critical protocol for reliable data transfer over IP networks. However, with the rapid growth of the Internet, many recent applications were designed for interactive use, in which delay is not tolerable. For such applications, TCP was found to be limiting because of its strict in-order delivery process. TCP is a

stream-based protocol suitable for activities carried over a long duration with data that need to be preserved. On the other hand, UDP is more convenient for transactions to be executed quickly and independently. It is not easy for applications that use both short and long transactions to develop a suitable trade-off that will result in an acceptable performance. In recent years, new transport protocols have been designed to provide proper support to different network applications. One possible design approach is to use *unordered* version of TCP (*u*TCP) as a base component and build application-level libraries on top of it [34]. A different design approach is to replace TCP with UDP and implement, at the application level, the required level of reliability. DCCP [35] is a protocol that followed this approach and was designed to provide only the congestion control mechanism to a datagram transport. However, most of these approaches have not been widely deployed or used so far.

One recent protocol following the same design approach and is being deployed and used by an increasing number of applications recently is Google's new protocol called QUIC. *Quick UDP Internet Protocol (QUIC)* uses UDP as the transport protocol to avoid the limitations of TCP and implements at the user level the congestion and flow control mechanisms.

QUIC was designed with the motivation of reducing communication delay introduced by the handshaking process and by the head-of-line blocking while providing an acceptable level of security and deployability. QUIC is deployed at the user space to enable its deployment across different platforms. To eliminate the head-of-line blocking issue, QUIC uses an abstracted data structure called *streams* and multiplexes multiple streams within the same connection. QUIC streams represent a reliable bidirectional communication byte-stream. Streams are uniquely identified by stream ID, and the

units sent over streams are called *frames*. A QUIC packet, as illustrated in figure 2.6 is composed of a header and one or more *frames*. After the early handshaking packet exchange, all QUIC packets are fully authenticated and encrypted except for the header parts required for routing and decryption. QUIC implements loss recovery, flow control, and congestion control mechanisms on top of the UDP implementation to ensure reliable transmission.

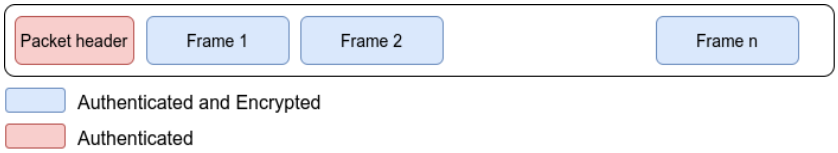


Figure 2.6: QUIC's packet structure

QUIC avoids the head-of-line blocking problem allowing multiple streams to be transferred over the same connection while ensuring that a lost UDP packet only affects the stream to which it belongs, while other streams can continue to deliver their subsequent packets. Moreover, QUIC limits the buffer space assigned to each specific stream.

Security Concerns

TCP protocol uses TLS for securing its traffic, which has been proven to provide solid authentication and confidentiality. On the other hand, QUIC protocol uses a different security library, which we will discuss later. We will explain how QUIC is providing authentication and confidentiality to its traffic. In QUIC, all packets are authenticated and encrypted, except for the early negotiation packets and the retry packets. The receiver is always authenticated, while the initiator authentication is optional. The authenticating certificate of the receiver party is sent in a *server-config-seg* at the initial

handshake phase. The initiator stores the *server-config-seg* received at the connection setup and uses it for later communication. The encryption keys are computed using *Diffie-Hellman(DH)* and are based on the information exchanged during the handshake phase. Tampering with the initial handshaking packets will lead to wrong values of the keys, a reset packet will be sent, unauthenticated and unencrypted, to indicate the failure of the connection [36].

TLS has multiple combination methods of authentication and encryption, the method that was found the most robust is *encrypt-then-authenticate*. In this method, the data is first encrypted then an authentication MAC is computed over the ciphertext. QUIC's data are placed in frames, which are also encrypted and then authenticated within the packets, as shown in figure 2.6, ensuring a similar level of robustness. TLS is vulnerable to denial-of-service (DoS) attacks, where the attacker imitates a large number of TCP connections and exhausts the server with a large number of handshake requests. By neglecting unauthenticated traffic, QUIC reduces the risks for DoS attacks. Both protocols are susceptible to attacks targeting specific cryptographic standards implemented on the receiver party, such as Bleichenbacher's attack on RSA [37].

Reinforcement Learning

The basic Reinforcement Learning (RL) model consists of a learning agent and an environment [38][39]. The agent is able to read the *state* of the environment and learns from interacting with it the optimal *policy* (π) to compute the *action* (a) that would maximize a *reward* signal. The problems solved using RL are sequential decision-making problems. At each time step t, the agent learns the environment's state s_t and decides on the action to take a_t. The state of the environment then transits to $s_{(t+1)}$, and

the agent receives a reward feedback signal r_t. The methods used to solve Reinforcement problems can be categorized into *model-based* and *model-free* methods. In model-free methods, opposite to model-based, learning is done without planing using trial-and-error techniques. In model-free methods, there is no planning of the environment changes based on the action taken. Model-free methods are better suited to solve problems where building a sufficiently accurate environment model is complicated.

Another classification of the RL methods based on what the agent learns classify the methods into *value-based*, *policy-based*, and *actor-critic* methods. The agent with a policy depending on the action-value function's estimation follows the *value-based method*. The value function is written as follows:

$$Q^*(s,a) = max_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2+\ldots|s_t=s,a_t=a,\pi}] \tag{2.1}$$

where γ is the discount factor and Q is the maximum sum of rewards discounted with the factor γ using the policy $\pi = P(a|s)$.

In the *policy-based method*, the agent learns a *parameterized policy* and uses it to compute the action without referring to a value function. The vector of policy parameters is referred to as $\theta \in \mathbb{R}$. Then, at time t, the probability of taking action a given the system state s and the policy parameters θ is written as follows:

$$\pi(a|s,\theta) = Pr\{A_t = a|S_t = s, \theta_t = \theta\} \tag{2.2}$$

The agent learns the policy parameters using a scalar measure of performance that the function $J(\pi)$ represents. The objective of gradient methods is to maximize the performance by updating the parameters vector proportionally to the gradient.

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta J(\pi) \tag{2.3}$$

where α_θ is a positive step parameter and $\nabla_\theta J(\pi)$ is the gradient of function $J(\pi)$ with respect to the policy parameters θ [40]. Methods following this general scheme are

called *policy gradient methods*, and if the method depends also on the learned value function, a vector of value weights $w \in \mathbb{R}^N$ can be used to include the learned value. The definition of the objective function $J(\pi)$ depends on the problem's nature. If the agent is continuously interacting with the environment uninterruptedly, the expectation of the average reward at each time step is used to evaluate the policies as follows:

$$J(\pi) = lim_{t\to\infty} E_\pi[r_1 + r_2 + r_3 + ... + r_t] \tag{2.4}$$

The agent is attempting to maximize the reward value over an *episode* starting with state s_0 and terminated at a specific terminal state. The expectation of the total discounted rewards starting at s_0 until time T, where the episode is terminated, is used to evaluate the policies as follows:

$$J(\pi) = E_\pi[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_0] \tag{2.5}$$

The third learning method is the *actor-critic method* in which the algorithm estimates both *policy* and *value*. The *(actor)* part is responsible of defining the parameterized policy and uses it to compute actions based on the state of the environment. The *(critic)* part then evaluates the defined policy and criticizes it based on the received reward value. The actor uses the evaluation results from the critic to modify the defined policy to generate optimal actions[39].

Deep Deterministic Policy Gradient (DDPG)

Deterministic policy gradient (DPG) algorithms [41] consider the *deterministic* policy $\mu_\theta(s)$. Deterministic policies have a model-free simple form that follows the action-value function gradient. Deep deterministic policy gradient (DDPG) algorithms combine the approach of actor-critic and the aspects of Deep Q Networks (DQN)[42].

DDPG algorithms are model-free with a simple actor-critic structure which implements these algorithms straightforward and easier to scale for more complicated problems. Q-Learning cannot be applied directly to continuous action-space since it will require the optimization process of action t at each time step, which is time-consuming. An actor-critic algorithm based on DPG will be a more suitable approach for continuous action-spaces [43]. The DDPG algorithms start by initializing both networks' parameters, the actor $\mu(s|\theta^{\mu})$ and critic $Q(s, a|\theta^{Q})$ networks. In DDPG, like Q-learning, a *replay buffer (R)* is used to learn the states in mini-batches, not online. R is initialized to hold the tuple (s_t, a_t, r_t, s_{t+1}) at each time step t and is used by the actor and the critic to sample uniform mini-batches from the buffer. At the beginning of each episode, the DDPG algorithm starts an action exploration process and obtains the initial observation of s. For all time steps within the episode, an action a is evaluated by the policy and is executed to observe the reward r and the next state of the system s_{t+1} and save the tuple $I(s_t, a_t, r_t, s_{t+1})$ to the buffer R. Random sample tuples are drawn from the buffer and used to update the actor and critic networks.

CHAPTER 3: RELATED WORK

In this chapter, we survey the existing proposals for enhancing the performance of the Tor anonymity network. We start with reviewing the work addressing the performance-limiting issues. We follow that with a review of the proposed techniques for circuit scheduling and traffic prioritization. We finally survey the path selection algorithms proposed to avoid inevitable drawbacks in the original design.

Enhancing Tor's Performance

Many research proposals were made to improve the Tor network's performance by addressing multiple design weaknesses. In the following, we review the work presented to address Tor's transport layer design, circuit scheduling mechanism, and path selection algorithm.

Transport Layer Design

One of the earliest proposals for an alternate transport design of the Tor network was the work of Liberatore [44]. The author proposed an extension for the basic specifications of Tor by building the circuits on top of Datagram Transport Security Layer (DTLS)/UDP. The proposed approach did not offer an alternative to the reliability and in-order delivery functionalities of TCP. The lack of reliability raised two main problems in Liberatore's design. First, the encryption done by Tor at the level of relays was done using the counter mode, in which each block being encrypted depended on the previous and next ones. Hence, in the case of a lost packet, the decryption process would not be successful. Second, the integrity check at both ends of Tor's communication is based

on the assumption that no packet will be lost. The extension was meant to be used in parallel to the original design of Tor. The control cells are sent over the TCP connection; only the UDP payload cells are sent over the UDP connection. Eventually, Liberatore's extension did not go any further due to its problems.

Later on, Reardon and Goldberg [45] proposed an improved design using TCP over DTLS. In this design, TCP is moved to the user level while using the Datagram Transport Security Layer (DTLS) to secure OR communication. Each circuit is assigned a separate user-level TCP connection. The reliability and congestion control are done hop-by-hop. However, the use of user-level TCP suffers from several limitations, such as CPU cost. UDP-OR is another approach to improve Tor's performance that was proposed by Viecco [46]. Viecco used UDP for the communication between the ORs only, while the end connections at the OP and exit are using TCP. Viecco's design simplifies the processing of packets at intermediate routers; however, it does not provide reliability and in-order delivery functionalities, affecting Tor's cryptography and integrity validation. The *head-of-line blocking* problem rises from the fact that if one packet is lost on one TCP stream, all other streams are blocked until this lost packet is being resent. To address this problem Nowlan, *et al*[47], introduced uTor. In uTor, Un-ordered TCP (uTCP) is used for communication between Tor's node and is protected by Un-ordered TLS (uTLS). This design allowed TCP to send any available data regardless of the lost packet event. This design adds to the application layer the additional cost of processing the packets, which affects the network's overall performance. The evaluation of this design showed a minor improvement in the performance. Another approach to use datagram protocol as a base for Tor's transport layer was proposed by Loesing,*et al.* [48], using a modified version of libutp library of Bittorrent.

However, the implementation was not mature enough to be evaluated against the performance of vanilla Tor. Alsabah and Goldberg[49] proposed the use of a single TCP connection per circuit. To provide security, they used IPsec and its Encapsulation Security Payload protocol. Although the performance enhancement of PCTCP was significant, the use of IPsec with Tor faces many challenges. Gopal and Heninger [50] in their Torchestra proposed to use two separate TCP connections between each pair of communicating relays. One connection is dedicated to light-weight traffic; the other connection is used for bulk traffic. Torchestra was not tested on a large enough network to better understand how it improved Tor's performance.

Circuit Scheduling and Traffic Prioritization

One of the earliest algorithms proposed to address the problem of cross-circuit interference in Tor was the work of Tang and Goldberg [51]. This work uses the value of the *Exponential Weighted Moving Average (EWMA)* for each circuit to identify the type of traffic it is carrying. Interactive circuits are identified by a smaller EWMA value and assigned a higher priority. Although this prioritization scheduling is effective, further experiments [52] showed that this approach is not fair to bulk circuits and affects their performance. Besides, it affects the performance of interactive circuits under light traffic loads. Moreover, this scheme does not address the scheduling limitations at the connection level. In [53], a detailed study of the sources of congestion in Tor was presented and showed that the kernel-level queuing time is the factor of highest impact on the total delay in Tor. This long kernel queuing delay happens due to the flawed connection-level scheduling scheme. The authors then proposed KIST, an improvement to the EWMA scheduler that considered the kernel socket status. Evaluation results

showed that KIST improved the performance of interactive circuits. However, the bulk circuit's performance was even worse than the original Tor case. From the fairness perspective, the authors of [54] presented an analysis of the fairness of Tor's circuit scheduling and proposed a *max-min fair* scheduling approach. This approach achieves high value on the fairness index. However, it does not differentiate between interactive circuits and other bulk circuits. In a later work, [55] a back-pressure-based transport protocol (BackTap) was designed for the Tor network that handled the fairness issues in Tor as well as the congestion control mechanism. The fairness between circuits increased by 20% on Jain's index using BackTap. However, this transport protocol adds additional feedback signaling to Tor's traffic which is not recommended. *mTor*[56] introduced an adaptive "pulling" scheduling algorithm that was designed to work with their multi-path Tor. However, the performance evaluation of mTor showed minimal improvement compared to Tor in interactive circuits and worse performance in bulk circuits.

In [57], Tortoise is presented to impose a universal limit that penalizes the users running bulk applications. The rate-limiting by itself does not provide much improvement to the performance of Tor. However, encouraging more users to run routers will increase the network's capacity and enhance performance significantly. Tortoise shapes the traffic by active throttling, and the authors recommend using it alongside the existing schedulers of Tor. *Traffic classification* is a pivotal operation to be executed before the scheduling process. Traffic classification informs the scheduler of the type of traffic carried by the circuits, helping the scheduler consider each circuit's QoS requirement. In addition to the method defined in [51], the work in [58] proposed a machine learning-based classifier to identify the traffic of the circuits as one of three types *(bulk transfer, interactive, and*

streaming circuits. Their work showed that the entry guard could identify the traffic type with accuracy reaching 90%.

Tor Path Selection

Improving Tor's path selection strategy can be achieved by informing Tor's client of the Internet routing status. Research proposals for improving Tor's path selection algorithm can be categorized according to their objectives into anonymity-based and latency-based. In anonymity-based researches, the main focus is to mitigate location-based attacks such as traffic correlation attacks and guard placement attacks. On the other hand, latency-based research concerns circuit latency, and its goal is to avoid congested paths.

Conflux [59] is a traffic splitting approach that considers the circuit latency. Conflux builds more than one circuit at the client end (OP) that intersects at the same exit relay. The client measures the latency of each circuit and splits the traffic among them accordingly. In [60], the authors presented a low-latency AS-aware approach for Tor's path selection (*LASTor*). LASTor clusters the relays that are geographically close and builds a graph with these clusters. Then for path selection, LASTor runs a *Weighted Shortest Path* algorithm on the clustered graph. LASTor improves the performance by avoiding circuitous paths, which in turn reduces unnecessary latency. However, the geographical clustering of the relays affects the network anonymity and degrades the entropy. Hence, LASTor introduced a tunable parameter to allow the users to choose their preferable trade-off between latency and anonymity. The authors of [61] presented *PredicTor*, a path selection approach that makes use of latency measures to avoid congested paths and selects shorter paths geographically. The path performance

is predicted using a Random Forest classifier. If the path is fast, the client proceeds to circuit construction; otherwise, the client has to choose a different path. Authors of [62] uses node latency to selects relays for a circuit and keep a list of the measured latency values for all relays at the client.

Reinforcement Learning Resource Allocation

The problem of optimally delivering traffic in a dynamic environment and with different scheduling priorities and data demands is fundamental in computer networks. This problem presents a complex decision-making task that has to be carried out online. Many research proposals addressing such a problem were motivated by the advances in artificial intelligence techniques, namely reinforcement learning (RL). A reinforcement learning-based scheduler was presented in [63] which was designed to adapt to the variations in cellular network IoT traffic dynamically. The authors designed the RL controller to maximize the High Volume Flexible Time traffic while maintaining the conventional traffic with minimal degradation. In the work of Jafari and Meybody [64], reinforcement learning was used to best serve the QoS requirements of multiple traffic classes under various conditions. In a system where resources are limited with many competing users, fairness is essential in the resource allocation process. A reinforcement learning algorithm was proposed by [65] to achieve fair resources allocation in fog networks. For dynamic systems such as heterogeneous networks [66], the use of *model-free* reinforcement learning approach is the best fit to learn the optimal policy from the stochastic properties of the networks. The state and action variables in heterogeneous networks are continuous, and a policy-gradient-based actor-critic algorithm is used to solve such problems. Similarly, in optical networks, an actor-critic resource allocation

(ACRA) [67] was proposed to improve the performance of resource allocation in optical networks, considering the complex relations between the network features.

CHAPTER 4: QUICTOR: ENHANCING TOR FOR REAL-TIME COMMUNICATION

In this chapter, we present our solution for Tor's performance limiting issues at the transport layer. We explore the use of a datagram-based protocol for the Tor transport layer. We first introduce our proposed design for the Tor transport layer using Google's QUIC protocol. We follow that with an explanation of the implementation and deployment of our QUIC library into Tor's code. Later we present our empirical evaluation of *QuicTor* in terms of performance gain and security.

Introduction

As we discussed earlier, Tor multiplexes circuits traverse any pair of ORs over a single TCP connection. This cross-circuit interference design hinders the performance of interactive applications by promoting issues such as *head-of-line blocking*.

The head-of-line blocking problem has been well studied in the area of router design. In essence, this problem stems from the conflicting requirements of multiplexing streams onto a single connection and preserving the ordering of the combined stream in case of failure or packet loss. Head-of-line blocking is an issue related to the use of reliable transport protocols such as TCP. As illustrated in figure 4.1, this problem happens when a specific TCP flow loses a packet and requires a re-transmission. All subsequent packets of these flows and other flows over the same connection are blocked until the lost packet is recovered. In the Tor context, we can map each stream on the connection to a Tor stream passing over a circuit, and the TCP connection is the one maintained between two ORs on this circuit.

As Tor becomes more popular, we will likely observe similar situations where a file

download stream shares a TCP connection with a web browsing stream. Moreover, when network links become more congested due to limited capacity, we expect congestion-induced packet losses to become more common, which will lead to more occurrences of the head-of-line blocking problem. Reardon, *et al,*[45] measured the effect of packet dropping on shared TCP Connection. Reardon's experiments concluded that multiplexing circuits over a single TCP connection adds unnecessary latency and significantly degrades the throughput.

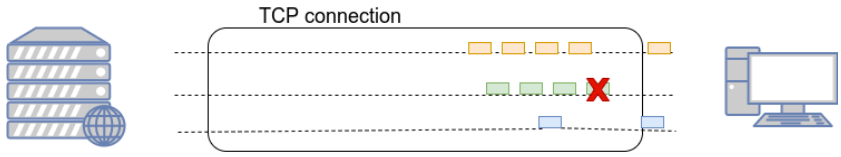


Figure 4.1: Head-of-line-blocking Problem Illustration

Approach and System Design

In our work, we built on the direction of using datagram protocol for Tor's transport layer, and we considered the problems faced by previous attempts. QUIC is a UDP-based multiplexed and secure transport protocol designed for bandwidth-hungry and latency-sensitive applications. Google designed it and now going through the standardization process in IETF standard track [68]. We choose QUIC because it was proposed as a standard way to address the head of line blocking at the transport layer in support of HTTP-2. We believe the same issues of the head of line blocking are affecting the performance of TOR. While research proposals such as PCTCP, IMUX, and Torchestra tried to solve the problem at the application level by proposing different methods to de-multiplex Tor's circuits, the use of UDP-based transport protocol, such as QUIC, would provide a solution at the transport-layer level which will also avoid the increasing

probability of socket-exhausting attacks. Moreover, the use of QUIC protocol for the transport design on Tor's network was considered by Tor's community as a promising direction to improve the performance of Tor [69]. We believe that QUIC is well-suited to address the two problems mentioned above. First of all, QUIC has native support for multiplexing multiple application-layer streams. This design allows QUIC to avoid the head-of-line blocking problem. Besides, QUIC has a pluggable congestion control module whose behavior is specific to each application-layer stream. This means that we can easily change its congestion control behavior for different circuits sharing the same connection in the Tor network. Figure 4.2 shows QuicTor's protocol stack implementation at each node along the path from the client to the destination. In QuicTor, all communication with onion routers uses QUIC, including the connection from the OP to the entry OR. To explain this design decision, we need to highlight some facts about QUIC and Tor traffic. QUIC traffic represents almost 7% of the overall Internet traffic[36], and it is never guaranteed that the destination server will be supporting QUIC protocol, as it is being adopted so far mainly by Google's services. Therefore, we kept the connection between the exit and the destination as it is. It can be seen that the TLS security layer in vanilla Tor was replaced by QUIC's security layer *QuicCrypto*. QuicCrypto is part of QUIC that provides transport layer security to a connection. The negotiation of used cryptographic suites is done during the cryptographic handshake, which QUIC combines with the transport handshake to reduce initial RTTs. Currently, QUIC is being drafted by IETF, and efforts are being made to move the cryptographic handshake implementation to be similar to TLS 1.3 [68]. Two essential works have analyzed the security of QUIC [70][71]. Both confirm it has reasonable security guarantees. QUIC/HTTP-2 was an inspiration work for TLS 1.3. Current versions of the

QUIC standard uses TLS 1.3 using creative designs to maintain QUIC performance advantages.

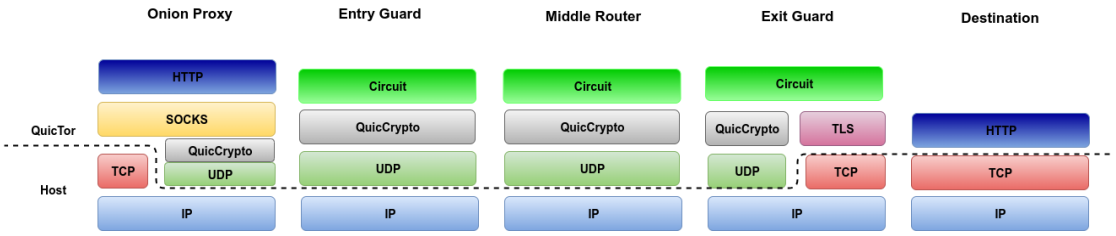


Figure 4.2: QuicTor protocol stack

To explain the QUIC communication process, we will refer to the two communicating ORs as *initiator OR* and *receiving OR*. As previously mentioned, QUIC's functionalities are implemented in user-space, including mechanisms to monitor events on UDP sockets and timeout alerts. This introduced a considerable challenge for our QuicTor API implementation to maintain an accurate timing method that would trigger QUIC's callbacks while dealing with the asynchronous events for the UDP sockets at user-space. QuicTor's API was packaged as a UNIX socket, which means that using a pooling loop to wait for socket events was not possible. To overcome this obstacle without significant re-writing of the code, a dedicated thread was generated for each UDP socket to process its events using libevent, while handling QUIC's alerts using libevent. The main thread communicates with each generated thread using a regular UNIX file descriptor (eventfd), which the user can treat like an actual socket.

When the initiator OR opens a connection, it starts a blocking operation to create a UDP socket and complete the handshaking with the receiving OR. Once the handshaking is complete, the main thread on the initiator OR generates a separate thread for this UDP socket to maintain the QUIC states' updates and the socket events. The generated thread will return eventfd that will be used to trigger the thread in case of pending reads. The

generated thread will be responsible for processing received packets without halting the main thread. On the receiving OR's end, the main thread will be listening for an incoming connection, creating a UDP socket, and generating a thread dedicated to this socket to handle its events.

One advantage of this design is that, since all libevent operations and QUIC states are handled in one thread, there will be no need for synchronizing multiple threads, reducing the implementation's complexity. For the few shared data structures, fine-grained locking is being used. A second advantage is that we provide a TCP-like usage by moving all asynchronous events to a background thread away from the main thread, in the same way the kernel is handling them for TCP. Finally, the interface for the API is a standard UNIX socket interface, which reduces the code changes to port existing Tor implementation.

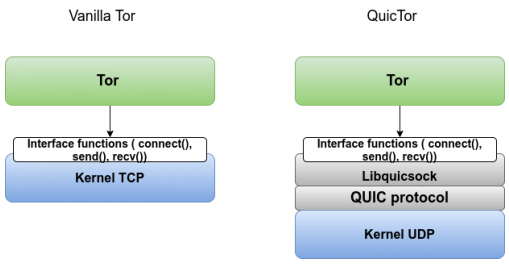


Figure 4.3: QuicTor Architecture

QUIC Deployment and Implementation in Tor's Network

Tor's original design layers the network communications as follows: connections and channels only describe the communication between two nodes. Circuits and streams, on the other hand, are end-to-end connections. A stream is maintained between the client and the server and is running on top of a circuit. At each hop, the circuit is mapped to a

connection. To use the QUIC API within Tor, we decided to limit the modifications to the connection layer while keeping other layers unchanged. Therefore, we added a flag to indicate whether the connection is *using quic* or it is a regular TCP connection. To allow incremental deployment and give the option of falling back to TCP at any point, we added a new QUIC socket to be used by quic connections along with the TCP socket created by Tor. The architecture of QuicTor is illustrated in figure 4.3 in which the different layers of QuicTor compared to the existing Vanilla Tor is shown. In QuicTor, the transport protocol used at the kernel layer is UDP. At the user-space level, the QUIC protocol implements its reliability and flow and congestion control functionalities.

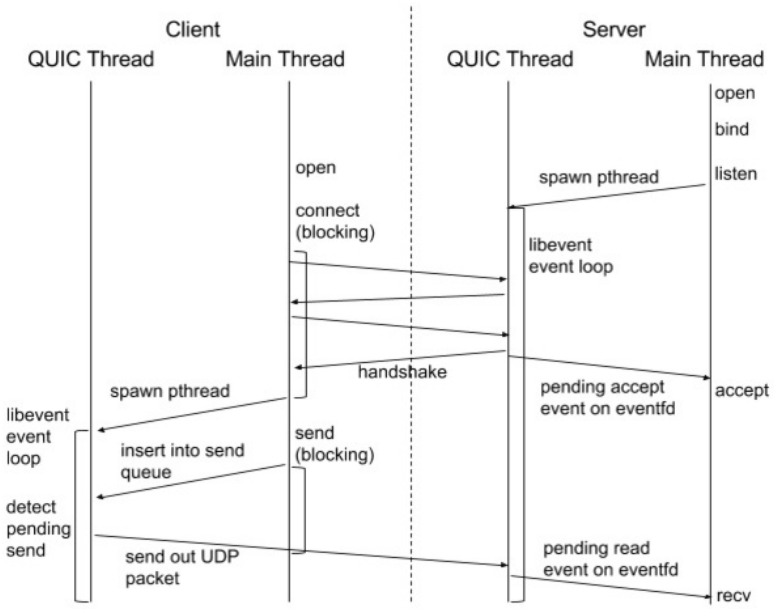


Figure 4.4: QuicTor API design

On the nodes that support the use of QUIC, all OR connections are being made using the QUIC socket, which includes OR-to-OR connections and OR-to-OP connections. We also simplified the connection layer read and write callbacks by transferring the TLS handshake process to QUIC, making it unnecessary to use the handshake code in Tor's callbacks. Other minor modifications that are not at the connection level were required

to support the use of QUIC. We needed to add a streamID field for the *packed_cell* structure to be used by QUIC to differentiate user streams. We used the *packed_cell* structure since it is the only one used by ORs for relaying the user streams. The streamID is used whenever Tor calls send to flush some packets to associate these packets with the correct stream. To avoid major modifications in Tor, we designed the QUIC library to provide a similar interface as TCP from Tor's perspective. The API functions connect, send, and recv follow TCP's blocking behavior while other functions are non-blocking. Moreover, in standard Tor, when a relay is about to send a cell, it will format the cell, copy it to the connection's output buffer, and add a pending write event to the event base. Then in the future, when the socket associated with the connection becomes writable, libevent will trigger the write event and run a callback function to send the data out. It is important to note that theoretically, QUIC has no notion of being writable as it uses a non-blocking UDP socket. This method means that Tor does not have to wait for buffer space since the buffers are all maintained by QUIC. However, to follow the TCP semantics, we decided to maintain this blocking behavior because we want to make sure that any performance gain comes from QUIC instead of changes in the semantics of TCP. See Figure 4.4 for an illustration of our design.

QuicTor Performance Evaluation

To show the performance gain achieved by the proposed design, we compare the presented work to two other proposed approaches that address the problem of circuit multiplexing over a single TCP connection. Namely, we compare our QuicTor to PCTCP [49] and IMUX [72]. However, PCTCP and IMUX address the head-of-line blocking problem at the application layer by de-multiplexing the circuits and use an appropriate

scheduler, while QuicTor addresses the problem at the transport layer. Figure 4.5 depicts the details of the OR-to-OR connections in QuicTor compared to Vanilla Tor, PCTCP, and IMUX. It can be perceived that QuicTor introduced minimum changes to Tor's architecture by merely adding a different socket identifier to be used for all OR-to-OR connections.

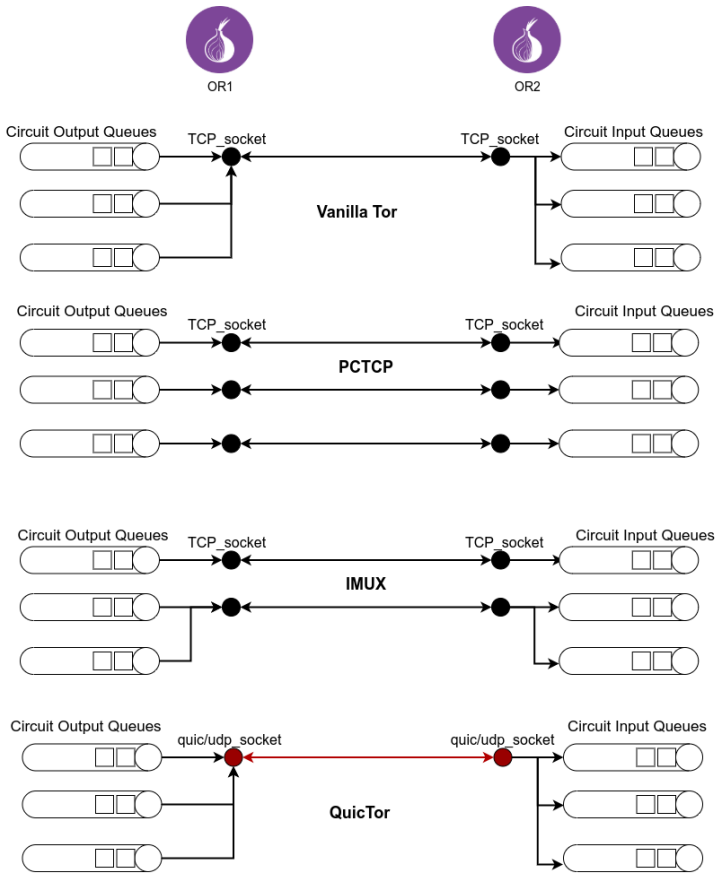


Figure 4.5: OR-to-OR connections in the different approaches

As a low-latency anonymity network, Tor aims to provide anonymity for the users of interactive web applications such as web browsing. Files downloading, e.g. using *BitTorrent*, is a commonly used application over the web that consumes plenty of its bandwidth [73]. We consider both types of applications in our evaluation of how QuicTor performs compared to vanilla Tor and different enhancement approaches, namely PCTCP

and IMUX. In recent years, video streaming has been the top internet application type in terms of traffic percentage. According to the report by Sandvin [73], video streaming reaches 58% of the global downstream traffic. Considering its importance, we evaluate the performance of video streaming over QuicTor compared to vanilla Tor. We implemented our design for QuicTor on Tor's source code version (0.3.3.5-rc)[1]. For a fair comparison, we ported the implementation of PCTCP and IMUX to the same version. We use a configuration flag to indicate which version of Tor is being used.

We set up our experiments using *NetMirage*[74] network emulator. NetMirage is a platform designed to allow testing IP-based network applications. The feature required in the tested application is the ability to bind to a specific IP address. We had to modify our code to pass the IP provided by NetMirage to the QUIC API for binding instead of using the *localhost* by default.

To ensure a fair comparison, we ported the implementation of both methods to the same version of Tor used in our experiment (0.3.3.5-rc).

Experiment Setup

NetMirage emulates the network on its *code node* using a GraphML file describing the topology of the network. NetMirage then generates IP addresses for the network nodes on its *edge node(s)* to be assigned to the tested applications. Traffic and communications between applications on the edge node(s) are routed through the core node. The network topology used for NetMirage configuration is in GraphML format, similar to the topologies used by other network simulators such as Shadow [52]. GraphML allows defining network parameters such as *latency, jitters, and drop rate*. In our experiments,

[1]QuicTor's source code is available upon request.

we configure NetMirage's core node using the model described by Jansen,*et al* [75]. NetMirage requires machines to run a Linux-based operating system. We used a machine with Intel Core i7 and 64 GB RAM that runs Ubuntu 16.4 for the NetMirage edge machine. We used an Intel Core i7 powered machine with 8 GB RAM running Ubuntu 16.4 OS for the core machine. We used a connected graph with each vertex representing a network node to configure the core machine of NetMirage. To simulate real internet behavior, we added latency to the edges that are randomly generated in the range of (50 ms - 100 ms) and drop rate in the range of (1% - 2.5%). The network configuration runs on the edge machine consists of 50 relays and 350 clients. 10% of the clients performed bulk downloads (files of size 5 MB), while the rest of the clients were sending regular HTTP requests representing web browsing activity. Conventionally, the web browsing activity is represented by the download of 320 KB files [49],[72],[76],[77],[78]. However, recently the average size of a web page increased drastically to reach more than 2 MB [79]. Hence, we used files of 2 MB in our experiment to represent web clients. We used a 5 minutes video uploaded on a separate server and dedicated one client for video streaming for the video streaming applications performance.

To validate the realism of our network, we used the performance metrics of Tor's live metrics [9] for 5 MB files, measured over the period starting from 01-11-2019 until 31-01-2020, to calibrate our configuration. The results of Vanilla Tor running on NetMirage's emulated network compared to Tor metrics are shown in figure 4.6. Tor metrics is an essential tool developed by *The Tor Project* to collect data of the live Tor network. The collected data is then aggregated, analyzed, and presented on the Tor metrics website [9]. Tor's relay performance is one of the metrics provided that researchers use as a reference for their experiments [49][56][77][80]. The emulated network using

NetMirage achieved a performance that is very close to the performance of Tor's live network with an average download time of 10 seconds for 5 MB files, which shows that our emulated network is realistic. We used this network for all of the performance evaluation experiments.

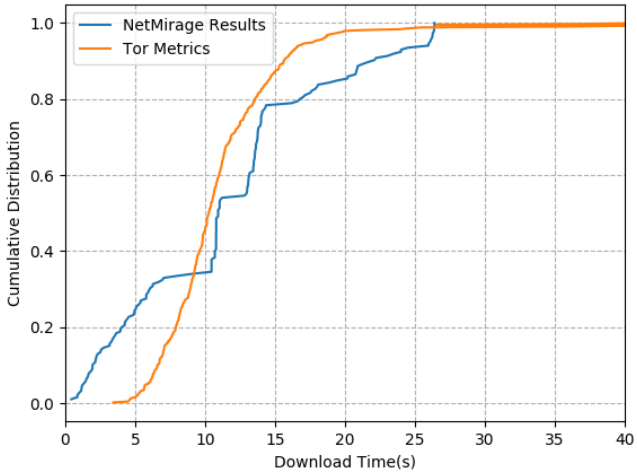


Figure 4.6: Tor network validation

Evaluation Metrics

For an application like web browsing and file downloading, the time required to complete the action, display the web content, or completely download the file is crucial in the network performance. We refer to this metric as *Download Time*. In Tor's experiments, the time required to establish the circuit and start receiving the first byte is an essential factor in its performance evaluation; it will be referred to later as *Time To First Byte*. We use both metrics to evaluate the performance gain of using QuicTor compared to Vanilla Tor.

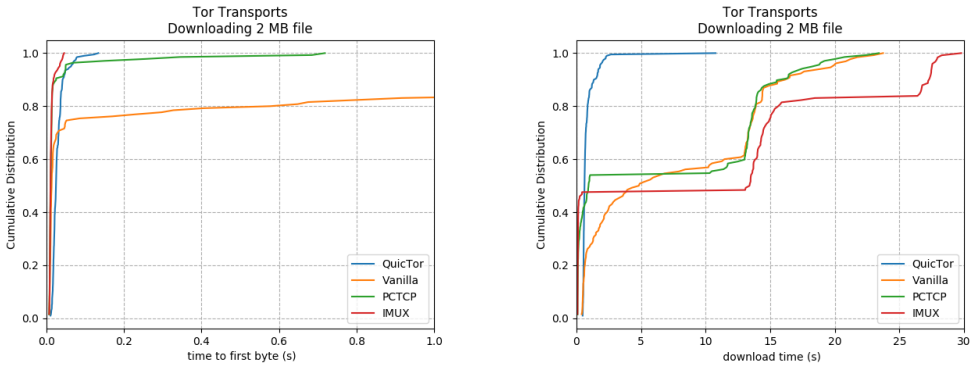
On the other hand, the user experience of different types of applications, such as video streaming, is measured by different metrics. In a study of how the quality of experience

(QoE) affects the user engagement in video streaming by Dobrian,*et al* [81], a set of metrics were described to evaluate the QoE for the video streaming applications. Out of the defined metrics, the following metrics are related to network performance.

- **Join Time:** The time required for the player to establish a connection, initialize the playing buffer, and fill the buffer to be able to start playing.
- **Buffering Ratio:** The buffering time as a percent of the total session time. Buffering time is the total time spent filling the playing buffer while the player is frozen.
- **Rate of Buffering Events:** The number of re-buffering events / total session time.

The *session time* is calculated as the total time since the client hits play until the end of the stream. We use these three metrics to evaluate the performance of streaming applications over QuicTor compared to vanilla Tor.

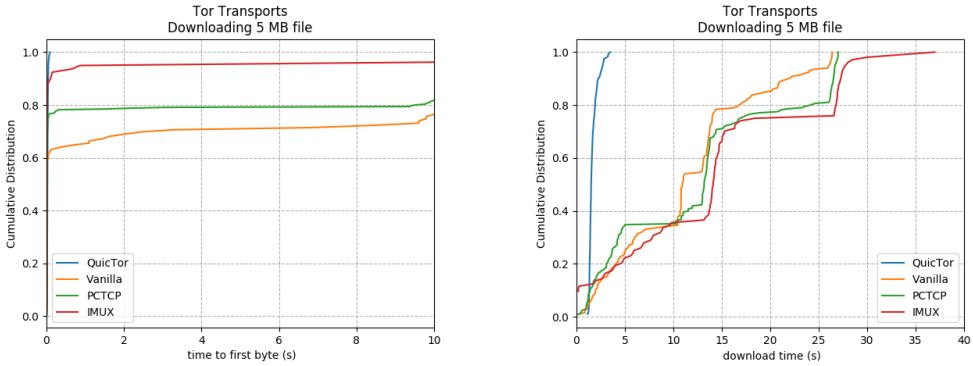
Results



(a) Time to download first byte

(b) Total Time to download file

Figure 4.7: Downloading 2 MB files



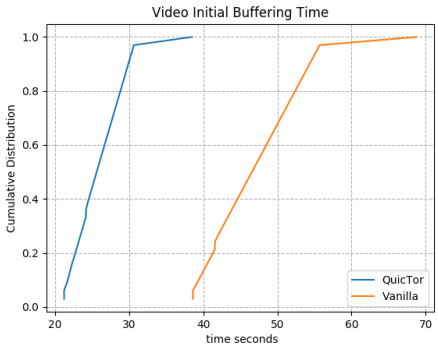
(a) Time to download first byte

(b) Total Time to download file

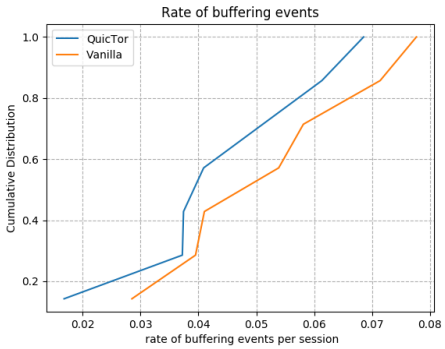
Figure 4.8: Downloading 5MB files

The main performance gain from the use of QUIC protocol instead of TLS/TCP lies in *reducing handshaking time* and *overcoming the head-of-line blocking problem*. The number of round trips required for handshaking is the primary source of pain for light-weight and short traffic such as web browsing. However, the actual performance gain, in this case, is minimal, equals two RTTs, and this can be shown by the Time to First Byte results, figure 4.7-a. In figure 4.7-b, it can be seen that 80% reduction of the average download time of a 2 MB file is achieved by using QuicTor compared to vanilla Tor. The average download time for PCTCP and IMUX is almost the same as QuicTor; however, The overall performance of using QuicTor is improved by 40% compared to PCTCP and IMUX. File sharing applications, on the other side, last for longer. Hence, they can benefit from the improved design of QUIC that eliminated the head-of-line blocking problem. In this case, the actual performance gain of QUIC can be noticed. Figure 4.8-a shows that 100% of QuicTor requests successfully established the connection in almost 1 second, while only 50% of Tor's connections were established within the same period. For the total time required to complete bulk file download, the

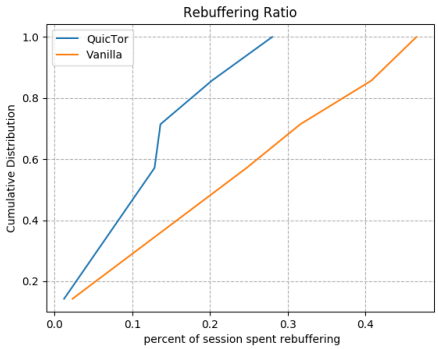
average for QuicTor is 3 seconds, and for vanilla, Tor is 15 seconds. QuicTor enhanced the performance for this type of application by almost 80%.



(a)



(b)



(c)

Figure 4.9: Video Streaming Performance Results

Video streaming applications also benefit from the reduced connection establishment latency of QUIC, which is reflected in the join time (initial buffering duration). It can be seen in figure 4.9-a that the average initial buffering duration in QuicTor is below 20 seconds, while for vanilla Tor, it exceeds 45 seconds. Figure 4.9-c and 4.9-b show two ratios that reflect the QoE presented to the user. The rate of buffering events represents how frequent the user will face a frozen player, the less this rate is, the better experience the user is getting. The rate of buffering events over QuicTor is 25% less than it is over

vanilla Tor. The second ratio is the buffering ratio, representing the percentage of the session time spent on buffering. QuicTor enhances this metric by 40% compared to vanilla Tor.

Security Analysis

In the following, we study different types of attacks with the goal of de-anonymizing the origin of Tor traffic. The goal of this study is to identify the attacks that could be affected by the use of a different transport protocol.

De-anonymization Attacks

Most of the de-anonymization attacks assume that the attacker is controlling at least one of the circuit hops, entry or exit guard, or both of them [82][83]. Furthermore, an attacker can present a compromised client or a malicious destination. An attacker can either passively monitor the traffic, or actively manipulate it. A global attacker can monitor the traffic end-to-end, Tor does not provide security against this type of attacker. A different assumption for an attacker is based on traffic monitoring. The attacker in this model can sniff the network packets and extract their features, train a model, and classify the traffic to identify it. The attacker can also manipulate the packets in a certain way [84] [85][15]. In Fingerprinting Attacks, the adversary is assumed to be able to monitor the traffic between the client and the entry point to the anonymity network. The adversary then extracts certain features from the traffic, such as packet count, flow direction, the time between consecutive packets. The next step is to match these features to indicative patterns of certain websites, using machine learning techniques. The effectiveness of these attacks depends on the selected features and the machine learning classifier used.

One of the earliest attempts to evaluate the effect of this type of attack on Tor's anonymity network was done by Herrmann,*et al*,[86]. The features they used were the frequency distribution of the size of IP packets, and the classifier used was multinomial Naive Bayes. Herrmann's classifier did not perform well on Tor since it only depended on the packet size, and Tor's cells have fixed size. Later, Panchenko, *et al*[87], worked on an enhanced version of fingerprinting attack on Tor by choosing different features based on the traffic volume, timing, and direction. Panchenko's classifier reached disturbing results raising red flags for Tor's community. Experimental defenses were recently developed against website fingerprinting attacks on Tor's anonymity network [88]. The AS-level attack is a traffic analysis attack enabled by the presence of the same AS network between the client and the entry guard and between the exit and destination. In their research, Edman and Syverson [89] provide an evaluation of the impact of the AS-level adversary on Tor network security. Their experiment showed that there is a probability of 20% that a single AS appears at the two ends of a circuit. This probability can be reduced by using a different path selection algorithm that is designed to avoid this problem.

Side Channel Attacks

Side-channel attacks are the type of attacks based on some information acquired about the network. In the context of Tor, side-channel attacks can be the first step to launch one of the previously discussed attacks by identifying ORs on the circuit. *Throughput Fingerprinting* is one of the attacks used for this purpose, it depends on the diverse nature of the volunteered routers and their unique behavior while building the circuit to identify the ORs.

Another type of side-channel attack aims to decrease the anonymity of the communication directly, such as *Network Latency*. Two network latency attacks were introduced by Hopper [90], the goal of the first attack was to identify the user initiating the traffic by analyzing the latency distribution of two exit nodes. The second attack aims to locate, approximately, the client by controlling a malicious server that collects any leaked information about the client's network every time the client tries to access the server.

With proposals being made to use datagram-based protocols for Tor's transport layer to improve its performance, an alarming security concern rises on how this type of protocol would affect the security and anonymity of Tor. The study was done by Mathewson and Perry [91] discussed thoroughly the different types of attacks, and specifically the attacks that are more likely to affect Tor over a datagram-based protocol. The described attacks in this study can be viewed as two main types. *First*, attacks exploiting protocol behavioral differences such as re-transmissions, congestion, and flow control. *Second*, attacks exploiting the reduced communication latency, such as timing correlations, and timing watermarking.

Various attacks were developed aiming to reduce the degree of Tor's network anonymity using different network performance metrics such as *latency and throughput*[92].

From the discussion mentioned above, we concluded that the category of attacks called *side-channel* attacks use some information gathered from the network traffic, such as delay and circuit lifetime. The use of different transport protocols could impact the nature of such information, which in turn would either facilitate or impede the launch of a side-channel attack on Tor's network. The attacks of this category can be *traffic correlation* attacks, or *traffic classification* attacks. In traffic correlation attacks, the adversary monitors the traffic at one end of the connection (entry/exit traffic) and one or more

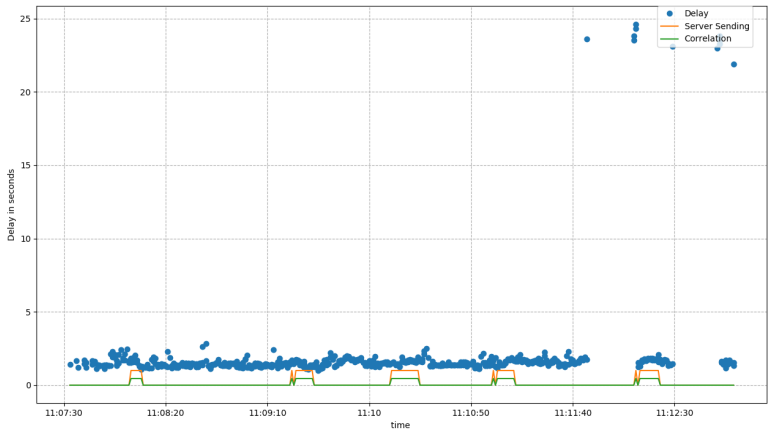
nodes within Tor's network. The adversary's target is to correlate the entry/exit traffic to the traffic monitored at one or more of Tor's relay to reduce the network's anonymity. To evaluate the security of QuicTor against side-channel attacks, we implemented two attacks, a timing-based attack described by Murdoch and Danezis [30] and a correlation attack described by Mittal,*et al,* [93].

Low-Cost Traffic Analysis of Tor

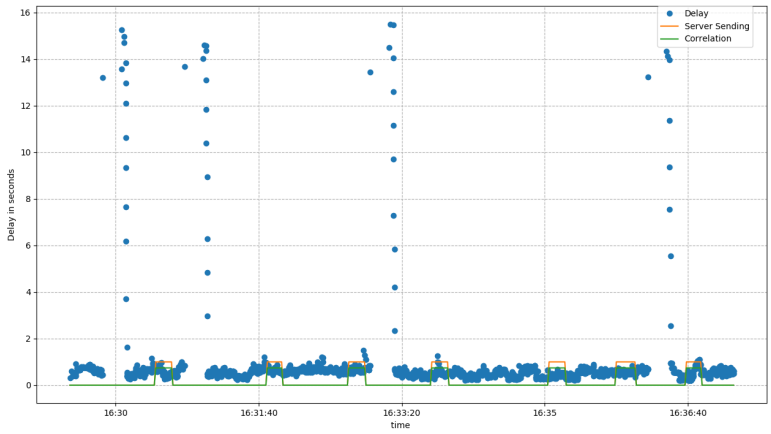
In this attack, [30] Murdoch and Danezis explained how an attacker could launch a traffic correlation attack despite the anonymity property that hides the direct link between communicating parties. Murdoch's attack is based on timing information that the adversary can acquire while staying within the threat model of Tor. The attack depends on the idea that traffic streams over Tor's network have specific characteristics and that a change in one stream can affect other streams passing through the same node. The adversary assumed in this attack is not global; he cannot observe the timing characteristics of the network. However, the adversary can inject his delay pattern into the network traffic and observe the network streams. The adversary is also assumed to control a corrupted Tor node, which is still within the threat model of Tor. To determine if the injected stream is passing through a specific Tor node, the adversary uses the corrupted node to send a stream to the targeted Tor node and measures the latency of this stream. The adversary then tries to spot the delay pattern injected by the corrupt server in the traffic of the probed relays and calculate a correlation percentage according to the formula :

$$c = \frac{\sum S(t) * L'(t)}{\sum S(t)} \tag{4.1}$$

Where S =1 if the server sends traffic at time t, and S = 0 otherwise. L'(t) is the normalized latency of the probed Tor relay. In a successful test, the correlation for a true positive (the injected traffic passes through the probed relay) should be higher than the correlation in the case of a true negative (the injected traffic does not pass through the probed relay).

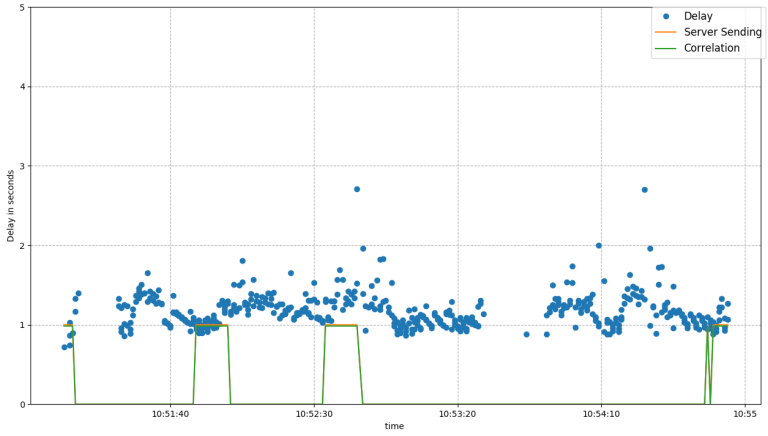


(a) True Negative

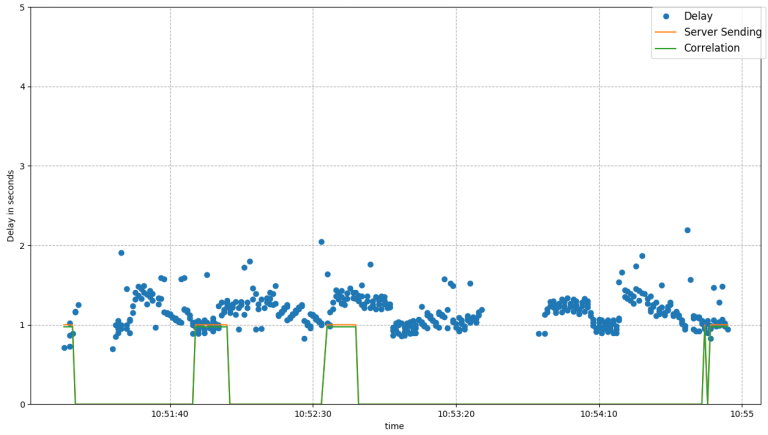


(b) True Positive

Figure 4.10: Probing Results of Vanilla Tor Relays



(a) True Negative



(b) True Positive

Figure 4.11: Probing Results of QuicTor Relays

We replicated the experiment described by the authors and tested for vanilla Tor to validate the original implementation. Then, we tried to launch the attack on the QuicTor network to evaluate its behavior against the attack. We created a network topology of 13 relays and 50 clients. One client, the one considered the victim, establishes a connection to the malicious server by creating a normal circuit of 3 relays. The malicious server keeps sending for a random period of 15-25 seconds, followed by a silent period of

20-40 seconds. A dedicated client is used and configured for each of the probed relays to allow a single hop. We bind a server to the same IP used for the probed relay and start sending through the client to that server and measure the latency. The rest of the clients are performing regular downloads over the network (web browsing applications). To validate our attack setup, we launched the attack against vanilla Tor and calculated the correlation value for both cases where the probed relay is and is not on the path between the victim client and the malicious server. Figure 4.10 shows the results from launching the attack on vanilla Tor. In 4.10-a the probing results of a relay that does not carry the injected traffic by the attacker, while in 4.10-b the probed relay is on the path from the corrupted server to the victim client. It can be seen from these results that the correlation value is higher in the case of true positive; this indicates a successful test and validates our setup.

The next step was to try launching the attack on QuicTor. Following the same process described for vanilla Tor, we obtained the results shown in figure 4.11. It was not possible to spot the injected delay pattern in the traffic from all probed relays, whether the relay is on the victim circuit or not. Using logs on QuicTor nodes, we identified the relays on the victim circuit and the relays that are not; the calculated correlation values were almost the same for both cases. The correlation value can be used as an indicator of the attack's impact on the network's anonymity. The value of the correlation between the probe data and the victim flow is higher in the cases where the pattern is present in the prob data. Using this information, the attacker can significantly reduce the anonymity set by considering the relays with correlation value $\geq$ a certain threshold T. In figure 4.12, we show the cumulative correlation measured for all probed relays. With correlation threshold $T = 0.4$ [93], it can be seen that the attacker can reduce the

anonymity set of the Vanilla Tor network to almost 25% of the total number of relays. On the other hand, the anonymity set of the QuicTor network was not affected. Using the entropy measures defined for measuring anonymity by [94], the attacker can reduce the entropy of the vanilla Tor network by 89%, while for the same correlation threshold, the attacker cannot confidently identify any of the relays on the circuits path.

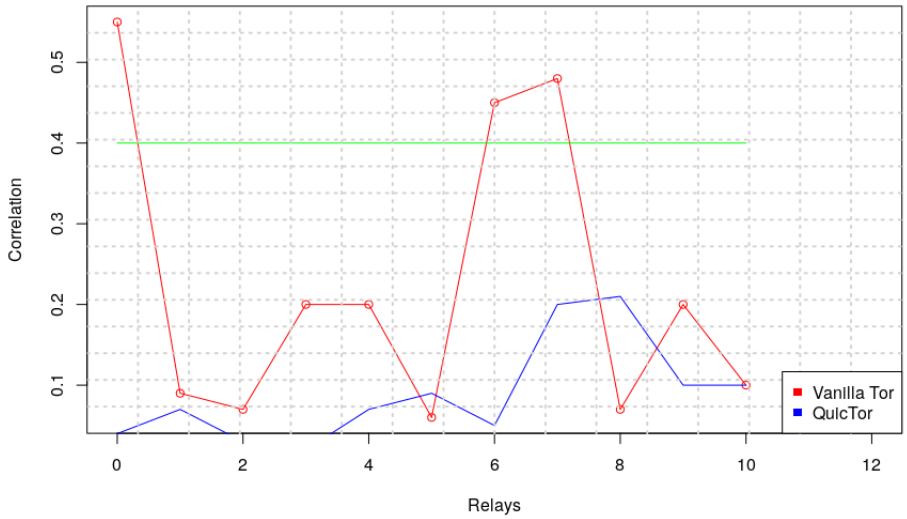


Figure 4.12: Correlation Measured

QUIC protocol uses a separate *stream* for every request/response sent to/from the server. Introducing delay in a certain server response will only affect the stream assigned to this response. Other streams for different requests/responses will not be affected by the introduced delay. When the attacker initiates a connection to probe the delay pattern of the relay in question, a new stream is created. The attacker stream, in this case, does not experience any additional delay. This makes it harder for the attacker to identify whether or not the examined relay is on the circuit path of the victim flow. Based on this, we can claim that timing-based attacks depending on tracking injected delay into the network can not successfully reduce the anonymity of QuicTor. A different attack that

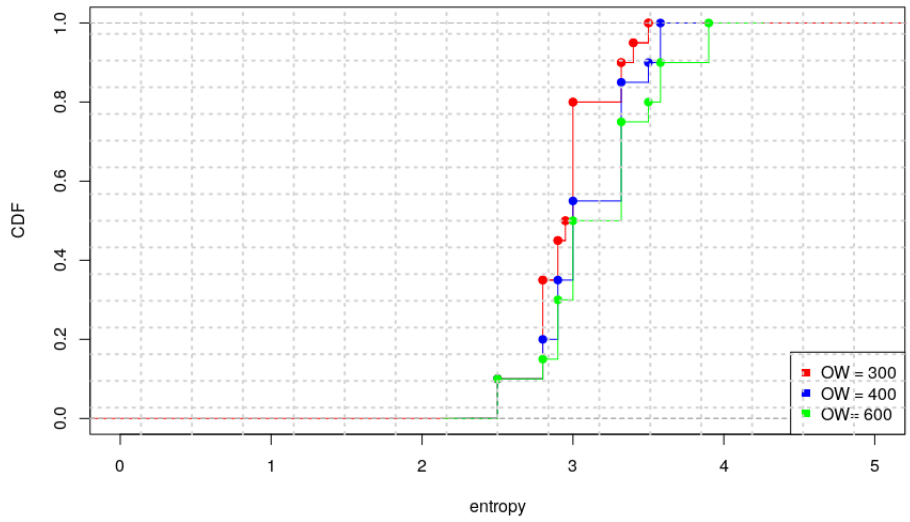
depends on injecting time gaps as a watermark was described by Iacovazzi, *et al*[95]. Iacovazzi's attack is a flow watermarking attack that aims to de-anonymize Tor's hidden services. Another flow watermarking attack that uses an inter-packet delay pattern as a watermark was introduced by Wang, *et al*[96].

Stealthy Traffic Analysis Using Throughput Fingerprinting

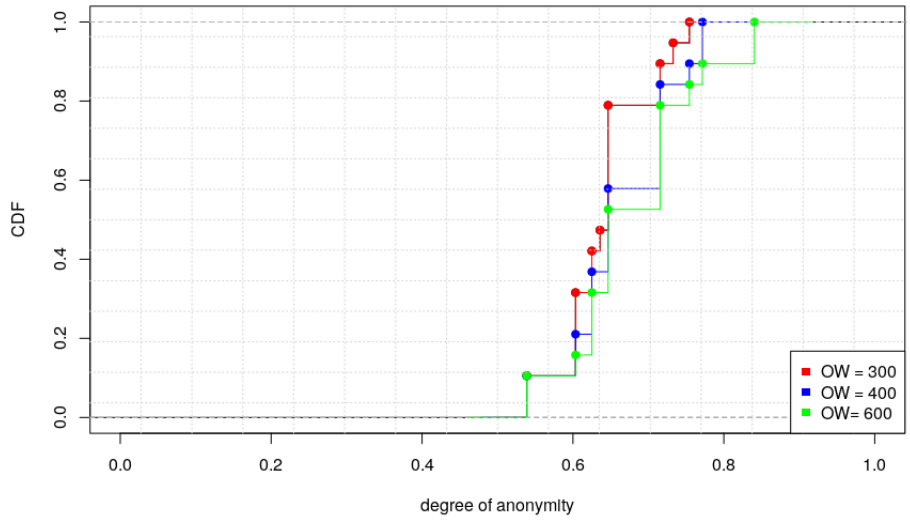
Mittal's attack [93] is a passive attack that does not require any altering or manipulation of the traffic. Instead, the attack uses the Tor flow's throughput as a fingerprint. The described attacker appears to be like any other Tor user, which makes it harder to detect that an attack is being launched. The authors described multiple scenarios to reduce the anonymity of Tor's network by implementing two types of fingerprinting, *stream-based fingerprinting and circuit-based fingerprinting*. *Circuit-based fingerprinting* is used to identify Tor relays, guard relays, and relays offering location hidden services. Mittal's work shows the correlation between the throughput of two circuits in different cases where the circuits share all three relays on the circuit path, two relays shared, and only one relay is common. A conclusion is drawn from these experiments that two circuits with highly correlated throughput have common Tor relay(s). To identify Tor relay(s) along the circuit path of the targeted (*victim*) flow, the attacker is assumed to be able to monitor the victim flow's throughput. The flow can be monitored by compromising the exit relay, the destination web server, or the ISP carrying the data. The attacker then probes the throughput of other relays in the network and tries to find a correlation with the throughput measured of the victim flow. To start probing the network relays, the attacker builds a one-hop circuit to these relays. The higher the correlation between the probe flow and the victim the flow, the more probable it that both flows are traversing

through a common relay.

We recreated Mittal's experiments, using 25 relays selected from the network topology we used to configure NetMirage in our previous experiments. We allow our attacker to observe the victim flow for an observing window (OW) of 300 seconds, 400 seconds, and 600 seconds; the observing window represents the lifetime of a client's circuit. In Mittal's experiments, they used a correlation threshold (T) of 0.4 that reflects the moderate confidence of the attacker. However, with QuicTor, none of the observed flows correlated higher than 0.3. Using this value as a threshold adds to the uncertainty of the attacker, which further weakens the attack. To quantify the degree of a system's anonymity, entropy is used as a measure [94][97]. *Entropy* is the level of uncertainty the attacker has about Tor relays in a circuit. After running the attack, the less the entropy is, the higher the probability of the attacker identifying the circuit relays. In Mittals' experiment, they reduced the entropy to less than 2.5 bits in 50% of the cases. Given that the maximum entropy for 25 relays is 4.6, the attack reduced the attacker's uncertainty by 40%. Figure 4.13 depicts the measured entropy after running the experiment for different observation windows. Only 10% of the cases were reduced to 2.5 bits, while 50% of the cases have entropy ≥ 3 bits. In 100% of the cases, the degree of the system's anonymity was ≥ 0.55.



(a) Entropy Set



(b) Degree of Anonymity

Figure 4.13: Entropy Set Reduction Results

CHAPTER 5: QDRL: QOS-AWARE CIRCUIT SCHEDULING

This chapter explains our proposed approach to achieve a balanced trade-off between efficiency and Quality-of-Service (QoS) delivered to Tor network users. We introduce a QoS-aware scheduling approach for Tor circuits queues. In the following, we discuss our proposed system model and problem formulation. We follow that with the approaches we used to solve Tor's scheduling problem. We evaluated the proposed approaches against the state-of-the-art scheduling method as well as some basic heuristics.

Introduction

In the context of Tor, *connections* are the TCP/TLS connection between communicating routers, while *circuits* are the logical end-to-end connections between the client and the destination. A circuit typically travels through multiple connections along its path. The nested hierarchy of the Tor scheduler is shown in figure 2.5. Each connection is linked to a buffer at the Tor application level. On the incoming side, received cells are stored in the incoming buffer until they are processed and mapped to the corresponding circuits. Similarly, cells are copied from the outgoing buffer to the socket on the outgoing side. A separate queue is maintained for each circuit to store cells belonging to it. The connection's outgoing buffer is filled from the queues of circuits traveling over the connection. The circuit-level scheduling was initially done using round-robin, later the traffic priority scheduler using EWMA was integrated into Tor's code. Tor uses the library called *libevent* to communicate with the kernel at the connection level. Each connection is mapped to a socket descriptor that is registered with libevent. Libevent notifies Tor asynchronously with the state of the connection if it is writable or not using callback functions. Libevent's notifications are triggered sequentially for one socket at

a time. Tor then tries to write as much as possible to this socket, not knowing about the state of other connections. KIST [80] addressed the sequential aspect of libevent by delaying Tor's response to the notification of libevent and collecting more information from the kernel before invoking the scheduling module. For the connection-level scheduling, the authors described two different algorithms to determine the *write limit* value (i.e., the maximum number of cells to be copied from the connection buffer at Tor level to the kernel socket buffer) to be imposed on *all* writable connections, which is the maximum number of cells to be copied from the connection buffer at Tor level to the kernel socket buffer. The circuit-level scheduling is not modified in KIST.

While each of the two scheduling levels in Tor attempts to achieve fairness, it does not

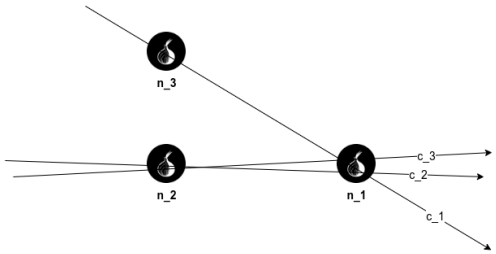


Figure 5.1: Tor Connection Network

reach overall fair resource allocation to all active circuits. A key aspect that leads to unfair allocation is that each scheduling level is handled independently. The assigned bandwidth limit for each connection does not consider the number and traffic types of the circuits mapped to this particular connection. To illustrate this major defect, we consider the simple connection from figure 5.1. This simple network model shows relay n_1 is maintaining two connections with relays n_2 and n_3. The connection between n_1 and n_2 carries two circuits while the connection between n_1 and n_3 carries only one circuit. In this scenario, assigning same bandwidth to connection n_1-n_2 and n_1-n_3 is not

really fair since circuit c_1 will use 50% of the total available bandwidth while circuits c_2 and c_3 will share the other 50% regardless of the priority of their traffic type.

Addressing these issues is not straightforward considering Tor relays' dynamic nature at each time step, such as the number of open connections, circuits mapped to each connection, and other parameters affecting the scheduling decision (e.g., the circuit activity and user status).

Approach and System Model

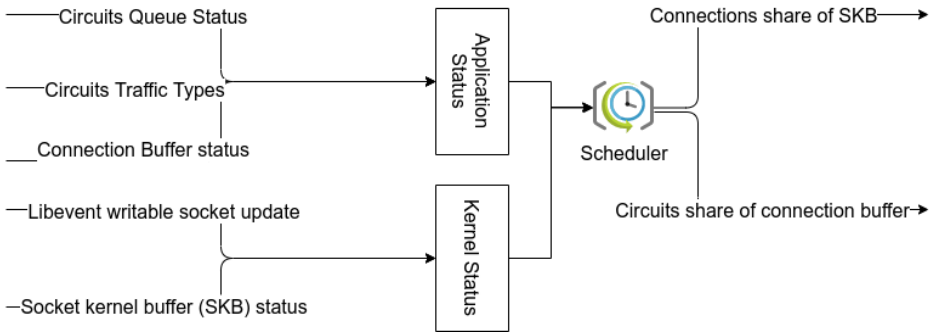


Figure 5.2: QDRL System Model

In our proposed system, we consider the two levels of scheduling as one scheduling process. The proposed scheduler for Tor relays is shown in figure 5.2. The system status received by the scheduler originates from two sources. First, the Tor application status describes the active circuits on each connection, the size of each corresponding queue, and the priority of each circuit based on its traffic type. Second, the kernel status gathers updates from *Libevent* on the writable connections and the writing limit to avoid overflowing the kernel buffer. The scheduler periodically reads the system updates, runs the scheduling algorithm, and allocates the resources based on the scheduling decision. The scheduler decides on the connections writing limit and the connection buffers allocation

scheme to active circuits. The scheduler's goal is to allocate the relay resources in a proportionally fair scheme while considering the QoS requirement for all active circuits. In the following, we describe three different scheduling approaches to achieve that goal: an optimization-based fair scheduling approach, average-rate proportionally fair (AR-PF) scheduling heuristic, and a reinforcement-learning-based scheduling approach. We end the section with a complexity analysis of the proposed methods.

Proposed Scheduling Approaches

This section discusses the proposed scheduling approaches to be used by our scheduler to achieve the desired weighted proportional fair allocation of Tor relay's resource.

Convex Optimization-based Fair Scheduling Approach

In convex optimization, the objective is to minimize a convex target function over a set of convex constraints. To obtain an optimal solution to the problem, we formulate it into a convex optimization problem. We formulate the problem as a convex optimization as follows. We start by defining a rate vector $r = \{r_1, r_2, ..., r_S\}$, where $r_j = \frac{(\lambda_j * q_j)}{\delta t}$ and S is the total number of active circuits in the system, λ_j is the percentage of circuit j's queue to be copied to the connection buffer, q_j is the total size of circuit j's queue, and δt is the observation window defined by KIST as 10 ms. We define a second vector $\omega = \{\omega_1, \omega_2, ..., \omega_S\}$ with the allocated resources for each connection corresponding to the active circuits. The priority of circuit $j \in S$ is referred to as p_j. Then, the throughput function for circuit $j \in S$ would be:

$$f = \omega_j * r_j * p_j \tag{5.1}$$

Function f is a non-convex function, however, the logarithm of this function ($log(f)$) is concave. This makes function f a *log-concave* function [98]. We define the system utility function to achieve the desired proportional fairness as follows:

$$U(\lambda, \omega) = \sum_{j=1}^{S} log(\omega_j * r_j * p_j) \tag{5.2}$$

Log-concavity is preserved under scaling and multiplications. However, the summation of log-concave functions 5.2 is not log-concave [98]. The utility function $U(\lambda, \omega)$ is a sum of logarithmic functions, which can be formulated as a log of multiplications using the logarithmic rule: $log\ a$ + $log\ b$ = $log\ ab$. The utility function $U(\lambda, \omega)$ will now look like this:

$$U(\lambda, \omega) = log(\Pi_{j=1}^{S}(\omega_j * r_j * p_j)) \tag{5.3}$$

We achieve a concave formulation of the utility function by applying exponential (e) to 5.3:

$$U(\lambda, \omega) = (\Pi_{k=1}^{S}(\omega_k * r_k * \gamma_k)) \tag{5.4}$$

The objective of our scheduler is to maximize the total throughput of the system. Hence, the utility function described by equation 5.4 can be put in the form of an optimization problem:

$$maximize_{\lambda,\omega} : U(\lambda, \omega) = \sum_{j=1}^{S} log(\omega_j * r_j * p_j) \tag{5.5}$$

$$\text{Subject to} \qquad \sum_{j}^{S} r_j \leq x_j \tag{5.6}$$

$$\sum_{j}^{S} \omega_j = 1 \tag{5.7}$$

At each time step, the solution of the problem 5.4 is the optimal trade-off between system fairness and QoS delivered to users. However, using convex optimization is not suitable for a large network setup due to the complexity. In the following subsection, we describe

a more scalable heuristic.

Average-rate Proportionally Fair (AR-PF) Scheduling Heuristic

Inspired by the concept in [99] used for designing proportional fair scheduler in communication networks through assigning network resources in a multi-user network-based on users' channel states, we propose a proportional fair scheduler in Tor network through dividing OR connection amongst different circuits using their queue state and circuit priority. For each circuit, we consider the average of the previous rate of packet writing. We first consider the simple case where a relay maintains only one TCP connection at a time over which C circuits are multiplexed. Recalling the definition of the network throughput as the objective utility function from equation 5.27. The heuristic defines at every time step t_k an instantaneous rate r_j for each circuit j ($\forall$ j $\in$ C) so that,

$$r_j(t_k) = \frac{q_j(t_k) * \lambda_j(t_k)}{\delta t} \tag{5.8}$$

The average rate for circuit j's rate is defined as:

$$R_i(t_j) = \frac{\sum_{k=1}^{j-1} q_i(t_k) * \lambda_i(t_k)}{t_{j-1}} \tag{5.9}$$

The objective of the proposed scheduling algorithm is to allocate the connection resources such that:

$$p_1 * \frac{r_1}{R_1} \approx p_2 * \frac{r_2}{R_2} \approx ..... \approx p_C * \frac{r_C}{R_C} \tag{5.10}$$

using the definition of r in (5.8), equation (5.10) can be written as follows:

$$\frac{\lambda_1 * q_1}{h_1} \approx \frac{\lambda_2 * q_2}{h_2} \approx ..... \approx \frac{\lambda_C * q_C}{h_C} \tag{5.11}$$

Where $h_j(t_k)$ = $\frac{\delta t * R_j(t_k)}{p_j}$

The constraint on the λ assignment is

$$\sum_{j=1}^{C} \lambda_j * q_j \leq x \tag{5.12}$$

The algorithm can calculate λ as follows:

$$\lambda_j(t_k) = \frac{h_j(t_k) * x}{q_j(t_k) * \sum_{j=1}^{C} h_j(t_k)} \tag{5.13}$$

At this point, we extended our heuristic to the general case of the Tor network where at any given time, a typical Tor relay will be maintaining N TCP connections, where N = the number of other relays it is communicating with. To avoid buffer-bloating, the writable connections should comply to a writing limit L such that:

$$\sum_{n=1}^{N} \omega_n * Z_n \leq L \tag{5.14}$$

where Z is the connection buffer size (at the application level) and ω is the percentage of the buffer to be copied to the kernel socket buffer (SKB). Following the same Average Rate-based Proportional Fair algorithm definition and applying it at the connection level, we define the rate at which a connection (n) will write to the kernel buffer at time t_j can be defined as follows:

$$\rho_n(t_j) = \frac{\omega_n(t_j) * Z_n(t_j)}{\nabla t} \tag{5.15}$$

Furthermore, the average writing rate for connection n will be:

$$P_i(t_k) = \frac{\sum_{k=1}^{t_{j-1}} \omega_n(t_k) * Z_n(t_k)}{t_{j-1}} \tag{5.16}$$

Following the proportionally fair allocation method defined in our heuristic, the system will try to divide the kernel buffer resources between connections so that:

$$\Gamma_1(t_k) * \frac{\rho_1}{P_1} \simeq \Gamma_2(t_k) * \frac{\rho_2}{P_2} \simeq ..... \simeq \Gamma_n(t_k) * \frac{\rho_n}{P_n} \tag{5.17}$$

where Γ is the weight or priority of the connection. We then define η as:

$$\eta_i = \frac{\delta t * P_i}{\Gamma_i} \tag{5.18}$$

Using 5.18 in 5.17 to solve for ω:

$$\omega_i = \frac{\eta_i * L}{Z_i * \sum_{i=1}^{N} \eta_i} \tag{5.19}$$

Recalling the definition of λ:

$$\lambda_j(t_k) = \frac{h_j(t_k) * x(t_k)}{q_j(t_k) * \sum_{j=1}^{C} h_j(t_k)} \tag{5.20}$$

Where x is the available space in the connection buffer at time t_j and it can be represented in terms of ω and Z :

$$x_i(t_j) = \omega_i(t_k) Z_i(t_k) \tag{5.21}$$

Now, equation 5.20 can be written as follows to reflect the relation between λ and ω

$$\lambda_j(t_k) = \frac{h_j(t_k) * \omega_i(t_k) Z_i(t_k)}{q_j(t_k) * \sum_{j=1}^{C} h_j(t_k)} \tag{5.22}$$

$$\lambda_j(t_k) = \frac{h_j(t_k) * \eta_i * L}{q_j(t_k) * \sum_{j=1}^{C} h_j(t_k) * \sum_{i=1}^{N} \eta_i} \tag{5.23}$$

The implementation steps of the AR-PF heuristic are depicted in algorithm 1.

Algorithm 1 AR-PF Algorithm

Initialize the values of λ and ω randomly

for *each time step* t **do**

update the values of x, q, and Z

for $i \leftarrow 1$ **to** N **do**

calculate the values of P using equation 5.16

calculate the values of η using equation 5.18

for $j \leftarrow 1$ **to** C **do**

Compute the values of λ using equation 5.23

end

end

end

Tor Reinforcement Learning-based Scheduling (TRLS) Design

TRLS attempts to address the previously mentioned multi-level scheduling problem in Tor using reinforcement learning techniques. Tor relays run for a long time, receiving traffic continuously from clients. The number of active circuits is constantly changing based on the network status. Using the DDPG algorithm will be the best fit for such a dynamic environment. In the following, we describe in detail the system's state and action spaces.

System States and Actions

A Tor relay receives a continuous flow of data cells from thousands of circuits traveling through the relay to the next hop on their paths. At any point of time t,

out of the total open TCP sockets on the relay, N sockets are writable. The state of Tor's scheduling system S is a continuous space that can be represented by the set $S = \{[n_0, n_1, ..., n_N], [Z_0, Z_1, ..., Z_N]\}$, where Z is the available space in the connection buffer at Tor level, and $n_i = [\{p_0, q_0\}, \{p_1, q_1\}, ..., \{p_{Ci}, q_{Ci}\}] \forall i \in N$, where C is the number of active circuits mapped to connection n, *active* circuits are open circuits with pending cells in their queues. Each circuit is described with its priority p and queue size q. Circuits traffic priority is assigned based on the application type. A basic traffic classification can be done using cell inter-arrival time [51][58], where interactive applications' traffic is usually bursty with significant gaps between small bursts of cells. Bulk applications, on the other hand, have continuous traffic with no pauses through the stream. Since Tor gets notified about the kernel status sequentially, the KIST scheduler, which has been part of Tor since 2017, collects the kernel information over a short period. We set t to be equal to the time interval used by KIST.

At every time step the agent's decision $A_t = [a_0, a_1, ..., a_N]$ where a_i is the decision for each connection $a_i = \{w_i, [\lambda_0, \lambda_1, ..., \lambda_C]\}$, where w_i is the number of cells to be copied from connection i ($\forall i \in N$) buffer to the kernel buffer, and λ_j is the number of cells to be moved from circuit j ($\forall j \in C$) to the connection buffer.

Algorithm 2 TRLS Agent

Initialize critic and actor networks using parameter vectors θ^{μ} and θ^{Q}

Initialize replay buffer R **for** $episode \leftarrow 1$ **to** Max_E **do**

Initialize a random action exploration process M

Receive initial observation state s_1 (circuits traffic type and queues size)

for $t \leftarrow 1$ **to** T **do**

Generate action a_t = $\mu(s_t|\theta^{\mu})$ + M_t

Carry out the action a_t, calculate the reward r_t from 5.31, and update the circuits

queue size accordingly(s_{t+1})

Save the tuple $(s_t, a_t, r_t, s_{t+1})$to the buffer R

Random minibatch with K transitions (s_i, a_i, r_i, s_{i+1}) is drawn from R

Let $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu\prime})|\theta^{Q\prime})$

Minimize the loss L to update the critic: $L = \frac{1}{K}\sum_i (y_i - Q(s_i, a_i|\theta^{Q})^2)$

Use the sampled gradient to update the actor policy:

$\nabla_{\theta\mu}\mu|_{s_i} \approx \frac{1}{K}\nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta\mu}\mu(s|\theta^{\mu})|_{s_i}$

Update target Actor and Critic networks:

$\theta^{Q\prime} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q\prime}$

$\theta^{\mu\prime} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu\prime}$

end

end

Agent Design

In TRLS, the decision of the action is *determined* based on the environment state. The action space is continuous by definition. Building an accurate model of the two-

level scheduling scheme for Tor relays is a challenging task; using a model-free learning algorithm will be a more feasible solution in this case. The agent's goal is to learn the optimal policy π to map system states to actions that maximize the reward value. The most common approach used is the greedy algorithm where $\pi(S) = argmax_a q_*(s, a)$. However, using this approach in a continuous action space is troublesome since it requires calculating the global maximum at each time step. A computationally simple alternative is to move the policy in the direction of Q's gradient. For the deterministic policy μ_θ equation 2.3 would be written as follows:

$$\theta_{t+1} = \theta_t + \alpha \mathbb{E}_{s \sim p^{\mu^t}} [\nabla_\theta \mu_\theta \nabla_a Q^{\mu^t}(s, a)|_{a=\mu_\theta(s)}] \tag{5.24}$$

$\nabla_\theta \mu_\theta$ is the gradient of the policy with respect to the parameters θ, and $Q^{\mu^t}(s, a)$ is the action-value function. Recalling the definition of the performance objective function J from equation 2.4, for deterministic policy $\mu_\theta : S \rightarrow A$, the performance objective is defined as $J(\mu) = E_\mu[r^\gamma|\mu]$, and the distribution of the discounted state is $p^\mu(s)$. The *deterministic policy gradient* can be formulated as follows [41]:

$$\nabla_\theta J(\mu_\theta) = \int_S p^\mu \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(a, s)|_{a=\mu_\theta(s)} ds \tag{5.25}$$

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s \sim p^\mu} [\nabla_\theta \mu_\theta(s)_a Q^\mu(s, a)|_{a=\mu_\theta(s)}] \tag{5.26}$$

We use the deep deterministic policy gradient (DDPG) algorithm for the TRLS agent since it provides a model-free actor-critic approach applied to continuous space.

Reward Function

The ultimate goal of our TRLS is to maximize the throughput of the relay while maintaining overall fairness for all active circuits, considering the QoS requirements for every circuit based on its traffic type. In the following, we formulate the reward function

that reflects the scheduler goal to the DDPG agent. In TRLS, we adopt a weighted proportionally fair approach with each circuit's priority p depending on the traffic type. The utility function that captures the required proportional fairness criteria in resource allocation is $U(f) = log(f)$ [100]–[102]. We define the throughput of each connection n_i as follows:

$$f = \sum_{j=1}^{C} \frac{(\lambda_j * p_j) * q_j}{\delta t} \tag{5.27}$$

, where λ_j is the resource share assigned to circuit j. The allocated share of the kernel buffer to connection, n_i defined as ω_i and the connection buffer capacity available is x_i, and we can write our target utility function as:

$$U(\omega, \lambda) = \sum_{i=1}^{N} \sum_{j=0}^{C} log(\omega_i * \frac{q_{i,j} * p_{i,j} * \lambda_{i.j}}{\delta t}) \tag{5.28}$$

$$\text{Where} \qquad \sum_{j} \lambda_{i,j} * q_{i,j} \leq x_i \quad \forall i \forall t \tag{5.29}$$

$$\sum_{i} \omega_i = 1 \tag{5.30}$$

The agent's goal is to maximize the value of U within the limitations described for λ and ω. At every time step, the agent decides the values of λ and ω, and based on this decision, we calculate the value of U and use it to reward the agent. However, the assignments of λ and ω that violate the restrictions defined should be punished. The *reward* function at time step t, r_t is defined as follows:

$$r_t = U(\lambda, \omega) - \beta_1 * (\sum_{i=1}^{N} x_i - \sum_{j=1}^{C} \lambda_{i,j} * q_{i,j}) - \beta_2 * (1 - \sum_{i=1}^{N} \omega_i) \tag{5.31}$$

,where β_1 and β_2 are tuning parameters. The reward function's negative terms represent the agent's punishment on an inaccurate assignment of the decision variables $[\lambda, \omega]$.

Performance Evaluation

This section presents a set of scheduling methods implemented as a baseline to compare the proposed scheduling techniques.

Simulated-KIST

KIST has been used for Tor scheduling since 2018, making it the first technique to be considered for our performance evaluation. In the first step of its scheduling process, KIST sets a writing limit for each writable connection. The writing limit is set using either a default value (*SIZE_MAX*) or based on the retrieved kernel information (*tcp buffer space*). In both cases, the writing limit is the same for all writable connections. At the circuit level, KIST uses the existing EWMA-based scheduling implemented in Tor. For the purpose of performance evaluation, we simulated the KIST scheduling algorithm's behavior where the writing limit for connection i; $\omega_i = \frac{1}{N}$, where N is the total number of writable connections. Next, for each circuit, we calculate the cell count value (v) at time step $t + 1$ as follows:

$$v_{t+1} = v_t * 0.5^{\frac{\delta t}{H}} \tag{5.32}$$

H is a pre-defined parameter called *"half-life"* after which the previously calculated average is reduced to half. After calculating v for all circuits, the relay chooses the circuit with the least v.

Heuristic-based Scheduling Approaches

We further propose two simple heuristics to allocate Tor relay's resources.

1. *Random:* under this heuristic, we assign an equal writing limit (ω) to all writable

connections. Out of the active circuits mapped to each connection, the scheduler selects one circuit (c_j) *randomly* to copy its cell queue to the connection buffer. The assignment of circuits share of the buffer $\lambda_i \; \forall i \in C$ will be as follows:

$$\lambda_i = \begin{cases} 1 & \text{if i = j} \\ 0 & \text{otherwise} \end{cases} \tag{5.33}$$

2. *Priority-based:* similar to the random heuristic, we assume equal writing limits to all connections considered writable at time t. However, the priority-based heuristic computes the rate for each active circuit based on its assigned priority p. For a circuit c_j, the allocated buffer share

$$\lambda_j = \frac{p_j}{\sum_i^C p_i} \tag{5.34}$$

We use the definitions of λ from equations 5.33 and 5.34 to calculate the value of the throughput function f (5.1).

Experiment Setup

In this section, we describe the setup of the simulation environment we used to evaluate the fairness and efficiency of the proposed scheduling approaches. We start by describing the simulated network topology, followed by explaining how we represent different traffic types. We further explain the training process of the TRLS agent.

Network Setup

A methodical approach for building a Tor network model and simulating a realistic environment for experimenting was presented in [76]. As depicted in figure 5.3, the used topology is a fully connected network graph where the edges are the TCP connections

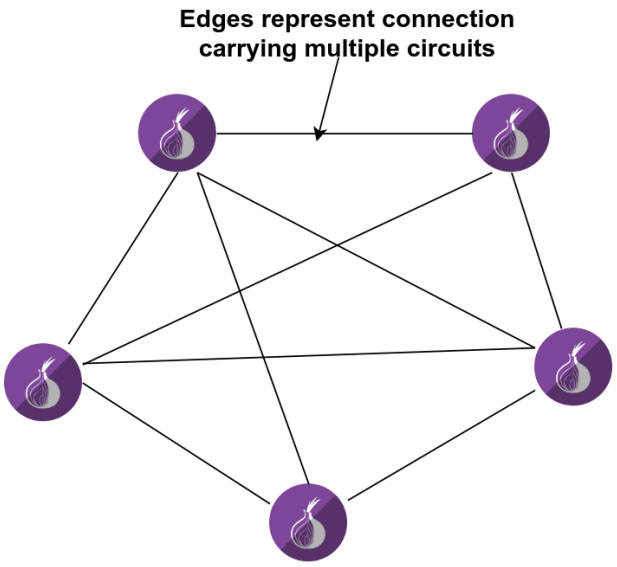


Figure 5.3: Network Topology

between relays. In a small network configuration of 50 relays and 500 clients, each relay maintains 50 TCP connections to all other relays. In Tor's live network, the number of relays is less than 30% of the total number of clients, which means a connection would carry hundreds of circuits at a time. In our simulation, we set the number of writable connections to be 5 connections with 20 active circuits per connection at a time. Circuits are assumed to be carrying one of three application types, *interactive web applications*, *bulk applications*, and *media streaming applications*. Interactive applications are assigned the highest priority, while the lowest priority goes to the bulk application since it consumes most of the resources. Interactive web application traffic, such as web browsing, can be simulated with bursts of data packets separated by long pauses. The size of these bursts is in the range of 4MB up to 6MB, corresponding to the recently reported average web page size[103]. On the other hand, bulk applications have continuous traffic streams and generally download files of size $>$ 50MB.

We simulate the kernel socket buffer restrictions by imposing a limit (L) calculated

based on the Linux kernel information. After each observation window, we simulate the flushed portion of the socket buffer and update the kernel buffers capacity using the average advertised relay bandwidth information [9]. We used the Python convex optimization solver (*CVXPY*) to solve the convex-optimization-based problem and the Python Keras DDPG library for agent implementation.

Table 5.1: TRLS Agent's Parameters

Parameter	Value
Episodes	600
Episode Length	100
Discount factor (γ)	0.995
Soft target update parameter τ	0.01
Actor network layers	3
Actor network learning rate	0.001
Critic network layer	3
Critic network learning rate	0.001
Replay buffer (R) size	10^6
Batch size	64

TRLS Agent Training

To train the RL agent, we configured the basic networks to train the agent for 3000 episodes corresponding to the default 30-sec *circuit lifetime* parameter of Tor circuits.

Each episode represents the observation window defined by KIST, 10 milliseconds. Networks learning rate is set to 0.001, and discount factor $\gamma = 0.995$. The agent contains two networks, the actor and critic networks. Each network consists of three layers; the input layer is a fully connected layer with 400 neurons. A batch normalization layer is used between the input layer and the activation layer, where *ReLU* is used for both networks' activation layers. We trained the DDPG agent using algorithm 2 with the

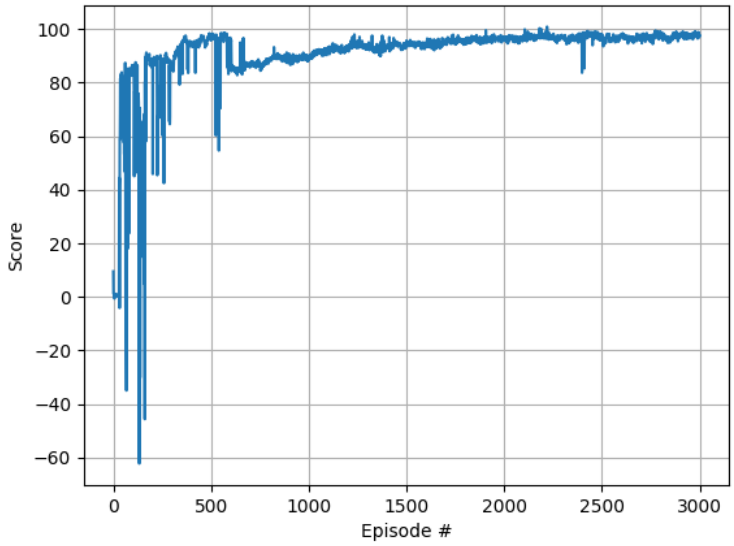
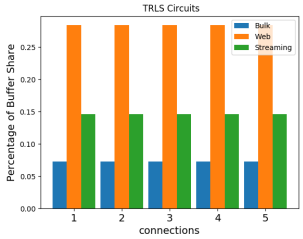
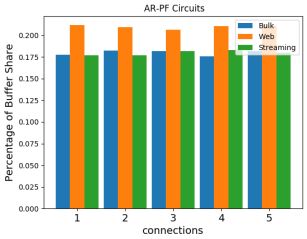


Figure 5.4: Agent Training

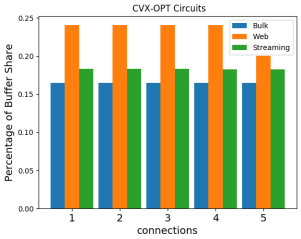
parameters values listed in table 5.1. The values of these parameters are set empirically based on the described model. In figure 5.4, we plot the obtained reward against the episode number. Over the first 1000 episodes, the agent explores policies to calculate actions that would maximize the reward within the system restrictions. The range of fluctuation in the reward value decreases as the agent keeps updating the target policies. By the 2500 episodes, the reward value reaches convergence around the value of 100. We used the trained agent for experimenting.



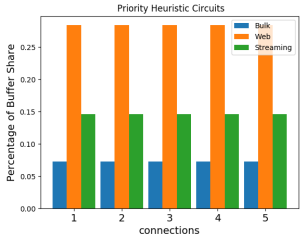
(a) TRLS



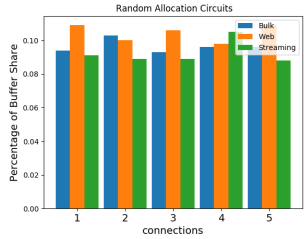
(b) AR-PF



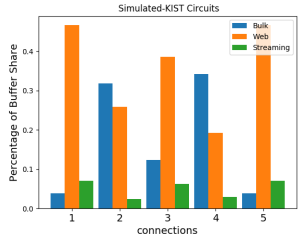
(c) CVX-OPT



(d) Priority Heuristic



(e) Random Allocation



(f) Simulated-KIST

Figure 5.5: Circuits Shares By Traffic Type

Performance Evaluation

In this section, we analyze and compare the performance of the three proposed scheduling approaches.

Quality of Service

Our primary focus in this work is to achieve overall proportional fairness while considering all QoS requirements. Figure 5.5 shows the assigned share of each connection to its active circuits categorized by traffic type. The reinforcement learning scheduling approach (*TRLS*) reflects the prioritizing system distinctly in the assigned buffers share to each circuit as depicted in figure (5.5-a). TRLS agent learns that the overall reward achieved increases as the assigned resources to the web browsing circuits increase. Bulk circuits, which represent bandwidth-consuming applications, are assigned the smallest

share of the connection buffer to prevent this type of application from affecting other interactive applications' performance. The Average-Rate Proportionally Fair (*AR-PF*) heuristic considers the current system status and the previously allocated rate for each circuit. As a result, circuits carrying web browsing traffic allocated a rate of only 15% higher than other traffic types. Streaming and bulk traffic are assigned almost equal rates. Allocated rates using AR-PF are shown in figure (5.5-b). At every time step, the convex-optimization-based scheduling approach (*CVX-OPT*) tries to optimize the system's throughput while considering the constraints of circuits priority. As a result, the buffer shares allocated to bulk and streaming circuits using this approach are higher than that using TRLS, as illustrated in figure (5.5-c). The resource allocation using the proposed priority heuristic mainly depends on the priority of each circuit. This concept is reflected on the circuit share of the connection buffer categorized by traffic type as shown in figure (5.5-d). The random allocation heuristic does not favor any circuit type; the decision process is done randomly. It can be seen in figure (5.5-e) that in some cases, the bulk circuits are allocated a higher share than the other circuit types. The KIST scheduler uses the EWMA method to categorize the circuits into light and heavy circuits based on their traffic characteristics. The scheduling decision is then made based on the average of the previously allocated value to each circuit. As a result, the web circuits receive the highest share compared to other types.

System Fairness

While the discussion above sheds light on how each approach handles the *QoS* requirements of different applications, we need to measure the fairness of the studied approaches quantitatively. For this purpose, we use *Jain's index(J)*[104] as a measure

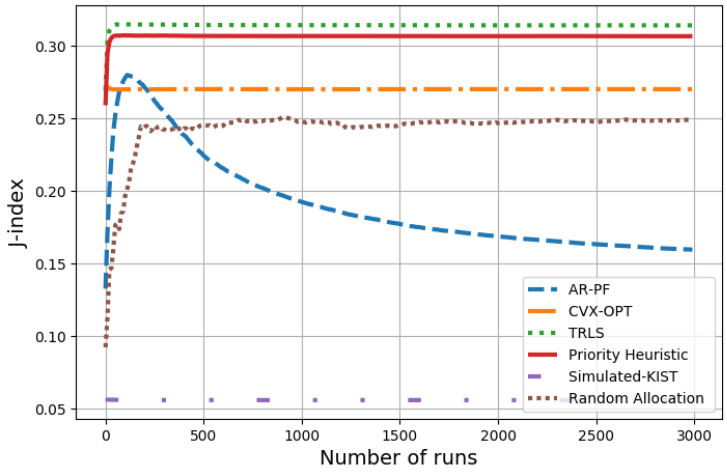


Figure 5.6: fairness of the studied scheduling algorithms using Jain's Index.

of the system fairness. Providing the same throughput to all types of circuits reduces the negative effect of bandwidth-greedy applications on the performance of interactive applications. Throughput-based fairness index is calculated as follows:

$$J(x) = \frac{[\sum_n T]^2}{n * \sum_n T^2} \tag{5.35}$$

n is the total number of active circuits, and T is the throughput achieved by each circuit. The index J indicates how fair is the schedulers to its users, and it has a value between 0 and 1. The higher the index, the more fair the resource allocation process is. For example, a system that assigns the same share of resources to all users will have an index = 1, while a system that assigns the larger portion of the resources to certain users while other users do not get any share will have a low index value. Figure 5.6 shows the discussed scheduling approaches' scores on the *j-index* scale. We measure the fairness index for all active circuits mapped to writable connections at time t. QoS-aware scheduling approaches allocate larger buffer shares to interactive circuits to achieve equal throughput to other bulk circuits. The results show that TRLS and the priority-based

heuristic scheduling approaches scored the highest fairness index. Simulated-KIST was shown to be the least fair scheduling approach in terms of the throughput of active circuits. The AR-PF approach considers the previous rates allocated to connections and circuits for future decisions; this approach reflects poorly on the fairness index score as it does not necessarily favor the interactive circuits. The convex-optimization-based approach is designed to maximize the throughput while considering the QoS requirements of different circuit types; this brings the CVX-OPT scheduling approach to third place on the fairness index, 15% less than TRLS. The random-allocation heuristic carries out the scheduling process randomly without considering the circuit type.

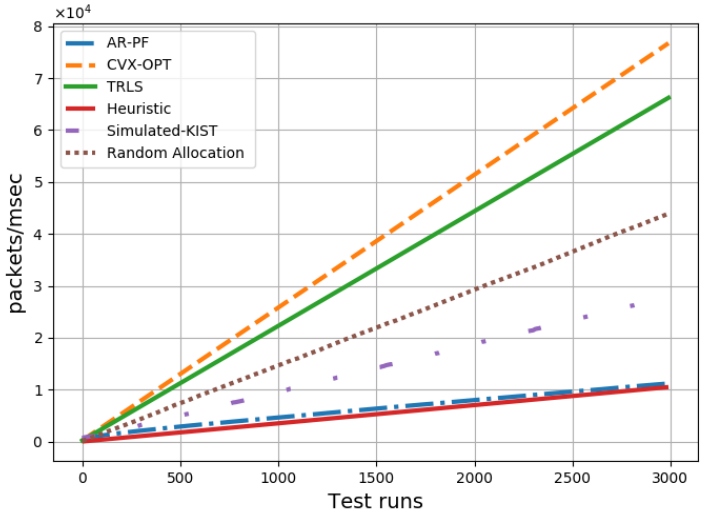


Figure 5.7: Throughput

Network Throughput

Throughput is an essential measure that reflects the efficiency of the system. In the following, we discuss the cumulative throughput achieved by the studied scheduling approaches. Designing a resource allocation scheme that is fair to its users does not

guarantee to achieve high throughput. In most cases, the higher the fairness index, the lower the throughput, and vice versa. Our experiment measured the number of packets flushed from circuit queues over the observation window (10 ms). Throughput results are shown in figure 5.7. The CVX-OPT scheduling approach aims to achieve maximum throughput within the constraints of QoS requirements, and it can be seen from the results that CVX-OPT achieves the highest cumulative throughput. Initially, the AR-PF assigns random rates for each circuit. Then, it reduces the rates based on the previous values to achieve the fairness objective. It can be seen from the results in figure 5.7 that after the first 1000 runs, the increase rate of the AR-PF cumulative throughput degraded. The TRLS scheduling approach maintains a steady growth of the cumulative throughput that is 20% less than CVX-OPT.

Network Delay

One of the limiting features of Tor is the delay users experience while using it. In the following, we discuss the impact of each scheduling approach on the network delay. Our experiment is based on simulating Tor's network behaviors under different scheduling approaches. Hence, the delay measurements are simulated results as well. In our setting, we set the test run time to be 30 seconds representing the default value of the *CircuitPriorityHalflife* parameter of Tor's relay, and the observation windows to be 10 milliseconds. To measure the delay of the circuit, we measure the number of windows required for a circuit to flush the average size of the web, bulk, or streaming request.

Figure 5.8 shows the cumulative distribution function (CDF) of the measured delay for the three types of circuits under the studied scheduling approaches. It can be noticed

that AR-PF, CVX-OPT, and TRLS have the shortest delay for interactive web circuits, while simulated-KIST has the longest delay. For circuits with streaming traffic, on the other hand, KIST performed significantly better. For bulk download circuits, the average delay using TRLS and KIST was 10% less than AR-PF and. 17% less thank CVX-OPT.

Live Tor Network Scenario

In this section, we investigate the performance of the studied scheduling approaches under the significant changes in the dynamic environment of the Tor network.

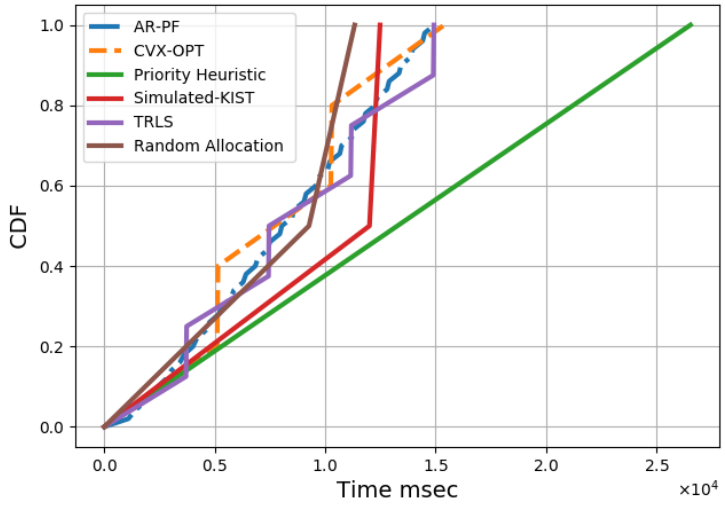
The number of users' active circuits traversing through the Tor relay is not fixed over time. New circuits are established in response to users' activities at any point in time. Moreover, circuits are not established simultaneously, which means they do not reach their lifetime limit and get destroyed simultaneously. Hence, during our observation window, the number of active circuits might increase or decrease according to the changes in the network state. In our experiment, we introduced the change in the number of active circuits at the midpoint of the observation window (after running for 15 seconds). In the following, we examine the effect of such changes on the network throughput and the fairness of the resource allocation approach used.

Based on Jain's fairness index calculation formula 5.35 [104], the value of the j-index is inversely proportional to the number of users, in our case, the number of circuits. Therefore, we would expect an increase in the j-index values if the number of active circuits decreased and vice versa.

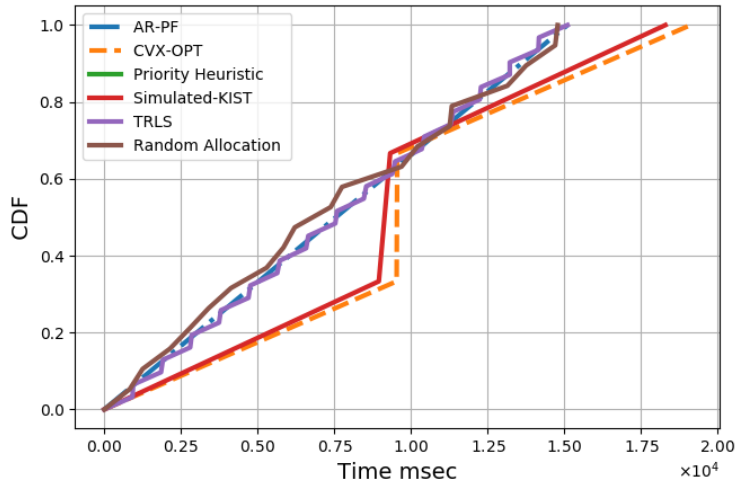
The fairness of the studied scheduling approaches after increasing the number of active circuits is depicted in figure (5.9-a). It can be seen that simulated-KIST fairness suffers the most from such change; it drops to half of its value and remains the same for the

rest of the test runs. The AR-PF heuristic will have to allocate random shares to the newly created circuits. Then, over the subsequent 500 runs (5 seconds), it recalculates the allocated rates based on its history and reaches a new stable fairness level. The fairness of the TRLS, CVX-OPT, priority heuristic and random-allocation scheduling approaches continue to rise over the subsequent test runs as the accumulated throughput increase. However, the new stable level of fairness remains less than the initial level. The throughput results, on the other hand, figure 5.9-b, show a noticeable reduction in the throughput growth of CVX-OPT and random-allocation while the TRLS throughput growth remains unaffected.

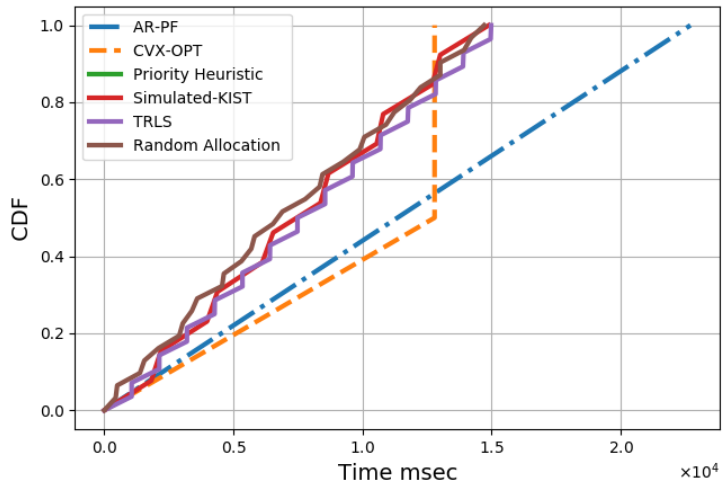
The effect of decreasing the number of active circuits on the performance of circuit scheduling approaches is shown in figure 5.10. It can be seen that the TRLS scheduling approach has the best response to the introduced change; once the number of circuits is decreased, the throughput drops for a brief period before the RL agent adapts to the new system state.



(a) Interactive Web Circuits

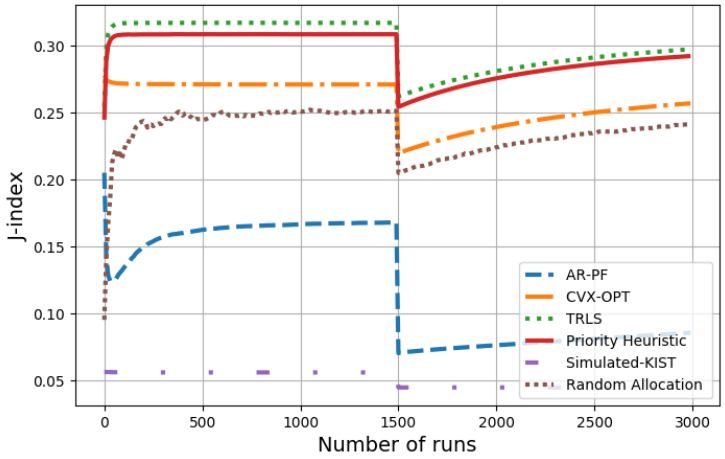


(b) Streaming Circuits

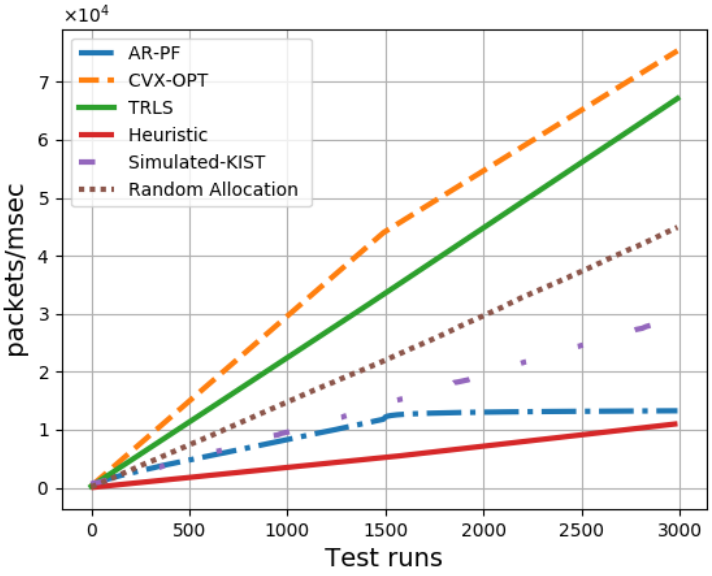


(c) Bulk Download Circuits

Figure 5.8: Delay Simulation Results

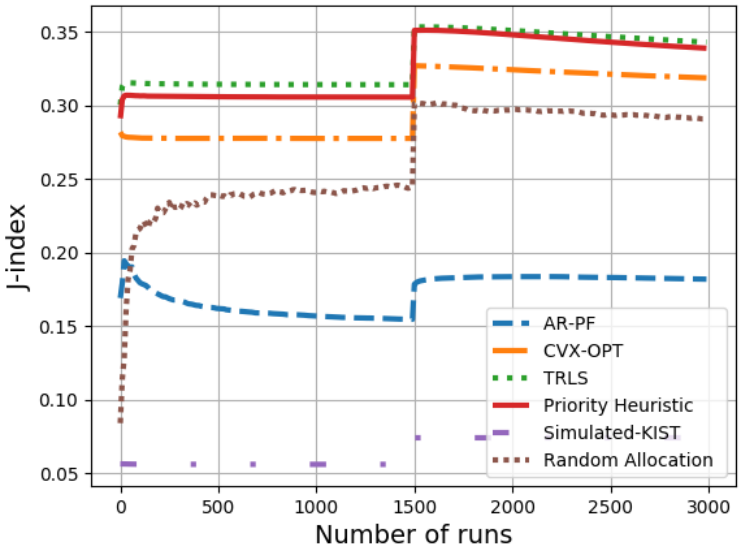


(a) Fairness Index

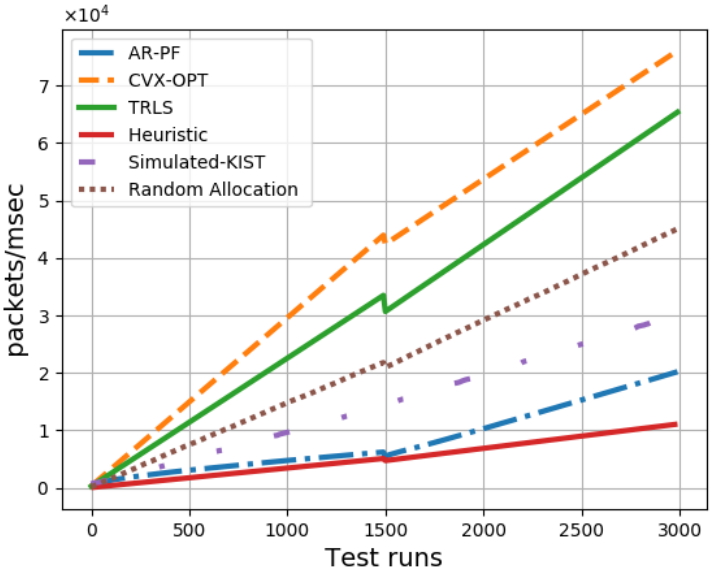


(b) Throughput

Figure 5.9: Number of Active Circuits Increased



(a) Fairness Index



(b) Throughput

Figure 5.10: Number of Active circuits Decreased

CHAPTER 6: FUTURE WORK

Future research work in enhancing the performance and security of Tor could be conducted by exploring solutions for other weaknesses in Tor's design or providing defenses against certain types of attacks. In our plans for future work, we follow two research directions; **First**, we address the path selection issues in Tor affecting the performance and the network anonymity. We started by studying the problem in the original path selection algorithm in Tor, and we formulated the path selection problem as a combinatorial optimization problem. Then we explored the use of the reinforcement learning approach to solve the problem in real-time. **The second** research direction is to address the problem of censorship and blocking of Tor traffic that limits the use of Tor in many countries. We explore the use of adversarial examples as a defense against traffic classification attacks.

Efficient location-aware path selection approach for Tor network

This section presents our proposal for a path selection algorithm that achieves the desired performance gain while preserving users' anonymity. We first formulate the path selection problem in Tor as a combinatorial optimization problem. Then, we propose our reinforcement learning-based model designed to achieve a decision on the selected path that reduces the overall latency while maintaining the users' anonymity intact.

Introduction

As mentioned earlier, a Tor client creates a circuit to relay the traffic to the destination over it. To build these circuits, the client needs to select three Onion Routers in an ordered sequence; entry, middle, and exit. The client then starts exchanging keys with these ORs

to be used for later cryptographic operations. ORs are selected proportionally to their measured bandwidth to increase the network throughput.

The challenging aspect of Tor's path selection algorithm is how to select the relays that would not introduce additional congestion in the network while reducing the probability of an adversary controlling more than one node on the path. Accordingly, the client chooses less congested relays and can not choose two relays with the same /16 IP addresses on any circuit path. In [13], the authors pointed out the effect of the relay selection strategy in Tor on its performance. The entry guards that have been used in the network for a long time tend to get overloaded, adding unwanted delay to the network. Moreover, the path selection strategy of Tor does not handle the load-balancing of the network in an efficient way.

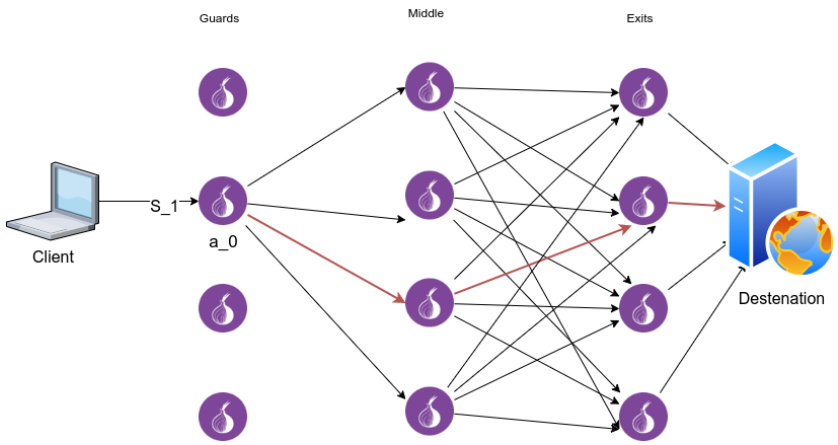


Figure 6.1: Tor Path Selection

Approach and System Model

Tor clients access the network through an *Onion Proxy (OP)*. The OP builds a virtual three-hop circuit incrementally, one hop at a time, following the above path selection rules. In the following, we present our formulation of the path selection problem and

the proposed approaches to solve it.

Problem Formulation

In this section, we formulate the problem of selecting the proper relays to construct Tor circuits paths as a combinatorial optimization problem. Tor network can be viewed as a graph (figure 6.1), with the vertices V are the onion routers (relays), and the edges E are the TCP connections between each pair of communicating relays. Tor's client is trying to find a path to the destination that consists of three hops that are not co-located. In other words, the client can not land on the same AS twice. The client would like to keep the cost of the path to a minimum. The path cost is the sum of the weight of the edges, which can be mapped to the relay latency. Relay latency can be measured using techniques similar to the one presented in [62], where the node congestion is computed as follows :

$$d = t - t_{min} + \gamma \tag{6.1}$$

where t is the measured *round-trip time*, t_{min} is the minimum *round-trip time*, and γ is a smoothing constant.

We define function $S(p)$ to be the cost function of path p.

$$S(p) = \sum_{i=1}^{n} d_i \tag{6.2}$$

where d_i is the measured delay of relay i, and n is the path length. The objective of the client is to minimize the cost while complying to the relay location constraints. We now define a function to compute the route from r_i to r_{i+1} as

$$route(r_i, r_{i+1}) = \{AS_1, .., AS_n\} \tag{6.3}$$

Formulating the problem subject to the location constraint, which states that no AS should appear on the path twice, can be done as follow:

$$\begin{aligned} &\underset{i}{\text{minimize}} \quad \textstyle\sum_{i=1}^{n} d_i \\ &\text{subject to} \quad route(r_i, r_{i+1}) \cap route(r_i, r_{i-1}) = \emptyset \end{aligned} \tag{6.4}$$

The larger the graph is, the more complex solving the combinatorial problem gets. In Tor's live network, nodes exceed 6000 relays, which requires a more efficient solution.

RL-TOPS: Reinforcement Learning-based approach for Tor Path Selection

In combinatorial problems, the solution is built iteratively based on the interaction with the problem's environment. In that perspective, a combinatorial problem such as Tor pas selection can be formulated as a sequential decision problem that can be modeled using *Markov Decision Process (MDP)*. MDP is used to describe an environment for reinforcement learning formally. The current state of the environment should fully characterize the process. The reinforcement learning agent decides on the optimal action to reach the maximum reward based on the system state. In the following, we describe the system states and actions for the path selection problem in the Tor network.

System States and Actions

At time t, the client OP considers a circuit path $c_i \in C$ of length L. While $L < 3$, the OP adds a suitable relay for the following position along the circuit path. P is the set of positions a relay can be used in to build the circuit, $P = \{guard(g), middle(m), exit(e)\}$. R the set of all network relays and R_p is the subset of relays that are suited for position $p \in P$. Each relay can be described by two main parameters; its bandwidth b and location l.

The system's state can be represented as a matrix of size C that holds for each circuit path the information of the relays appended to the path. At every decision epoch t, the state will provide a complete description of the system used by the agent to decide the optimal action. The action required at every epoch is selecting the most suitable relay to be added to the circuit path.

Reward Function

The goal of the reinforcement learning agent is to find the best path that would reduce the network congestion and reduce the probability of an adversary controlling more than one node along the circuit path. To this end, we use the following reward function to be used by the agent to evaluate the decision made at every epoch.

$$F = \alpha\omega_{dp}^{r} + \beta\omega_{l\,p}^{r} \quad \forall r \in R_p \tag{6.5}$$

ω_d^r is the delay weight of relay r and is computed using equation 6.1

ω_l^r is the location weight of relay r. Based on the route calculation we compute the location weight:

$$\omega_l^r = \begin{cases} 1 & \text{if} \quad route(r_p, r_{p+1}) \cap route(r_p, r_{p-1}) = \emptyset \\ -1 & \text{otherwise} \end{cases}$$

Agent Training

To train the RL agent, we configured the basic networks to train the agent for 4500 episodes. Networks learning rate is set to 0.00001, and discount factor γ = 0.995. The agent contains two networks, the actor and critic networks. Each network consists of three layers; the input layer is a fully-connected layer with 400 neurons. A batch normalization layer is used between the input layer and the activation layer, where *ReLU*

is used for both networks' activation layers. It can be seen from figure 6.2 that the agent reaches convergence after 2500 episodes and reaches an average reward value of 1.2.

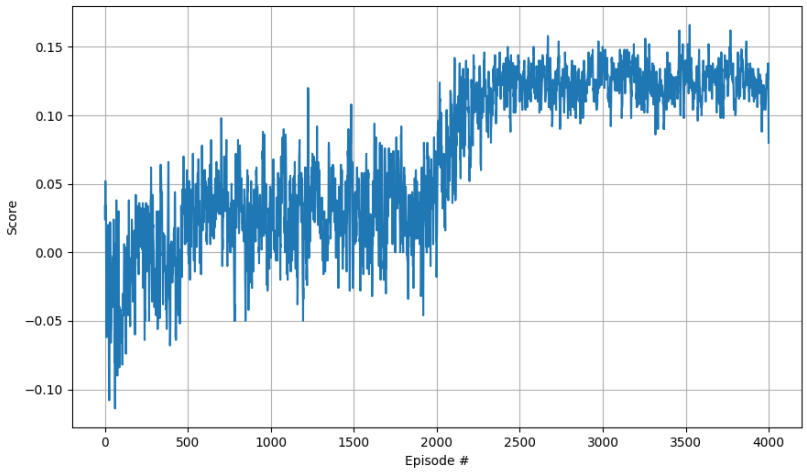


Figure 6.2: Agent Training

Defenses against Traffic Classification Attacks

While Tor relays can use traffic classification technologies to prioritize the circuit traffic, as we discussed in chapter 5, traffic classification can also be used by an adversary monitoring the traffic from the client to the entry guard and trying to differentiate Tor flows from non-Tor flows before blocking Tor flow.

The features used to train the classifier are either collected from the encrypted flow or by deep packet inspection. Traffic classification methods use different machine learning techniques for their classifiers, such as deep learning techniques. However, machine learning and deep learning techniques were proven to be vulnerable to carefully designed disruptions in the testing examples; these types of examples are referred to as *adversarial examples*. Adversarial examples force the machine learning algorithms to

perform poorly in terms of accuracy. The adversarial examples have been considered possible defenses against attacks that depend on machine learning techniques, such as website fingerprinting. We plan on exploring the possibility of using the adversarial example as a defense against traffic classification attacks on Tor entry traffic.

CHAPTER 7: CONCLUSION

In this dissertation, we discuss the importance of online privacy and the technologies used to preserve it. Low-latency anonymous communication networks are reliable tools for the users of interactive applications to protect their privacy. In this work, we highlighted three main weakness in Tor's design that results in poor performance. We presented several approaches that aim to improve the performance of the Tor network without compromising the anonymity of its users' communication.

We first explained the design of the transport layer of Tor and how it affects the network performance. Furthermore, we explained the QUIC protocol introduced by Google and discussed how it would reduce communication latency. In order to make use of QUIC's features to improve Tor's performance, we developed QuicTor; we first developed a user-level wrapping library for QUIC and modified Tor's code to use it for the connections between onion routers. We performed an empirical evaluation of our proposed design and compared the results to previously developed designs and the stock version of Tor. The results showed a significant reduction in the time needed to establish circuits and start downloading files; the improvement in the total download time reaches almost 80% for the bulk downloads. Given the accomplished improvement in performance, we plan to expand our network topology for a more realistic evaluation. Moreover, we plan to extend the assessment of the security and anonymity of Tor's network using the QUIC protocol.

We also addressed the problem of fairness in the Tor anonymity network. We described the design of a weighted proportionally fair scheduler for Tor's circuits. We reformulated the relay resource allocation problem in Tor to incorporate both circuit level and connection-level scheduling. The prime goal of the proposed scheduler is to achieve

the best trade-off between QoS-aware fairness and system efficiency. To achieve this goal, we introduced a set of solutions for a fair scheduler. *First*, we introduced a reinforcement learning-based scheduling approach (*TRLS*). We designed TRLS's agent to fit Tor's network's dynamic nature. *Second*, we formulated and solved Tor's circuit scheduling problem as a convex optimization problem (*CVX-OPT*). *Third*, we reformulated an average-rate-based proportionally fair heuristic (*AR-PF*) to consider the general case of the Tor network. We show that while CVX-OPT achieves the optimal system throughput, the proposed reinforcement-learning- based approach (TRLS) achieved the highest QoS-aware fairness level with a resilient performance to the changes in a dynamic environment, such as the Tor network.

PUBLICATIONS

- L. Basyoni, A. Erbad, M. Alsabah, N. Fetais, M. Guizani “Empirical performance evaluation of quic protocol for tor anonymity network”, 2019 15th International Wireless Communications Mobile Computing Conference (IWCMC). Best Paper Award
- L. Basyoni, N. Fetais, A. Erbad, A. Mohamed, M. Guizani, "Traffic analysis attacks on Tor: a survey," 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT).
- L. Basyoni, A. Erbad, M. Alsabah, N. Fetais, A. Mohamed, M. Guizani, "QuicTor: Enhancing Tor for Real-Time Communication Using QUIC Transport Protocol" in IEEE Access.
- L. Basyoni, A. Erbad, A. Mohamed, A. Refaey, M. Guizani "Proportionally Fair approach for Tor’s Circuits Scheduling", 2020 International Symposium on Networks, Computers and Communications (ISNCC).
- L. Basyoni, A. Erbad, A. Mohamed, M. Guizani "QDRL: QoS-aware Deep reinforcement learning approach for Tor’s Circuit Scheduling", IEEE Transactions on Network Science and Engineering (Submitted).

REFERENCES

[1] M. Ellis, *10 real examples of when data harvesting exposed your personal info*, May 2018. [Online]. Available: https://www.makeuseof.com/tag/data-harvesting-personal-info/.

[2] *Protecting human rights on the internet*, Mar. 2017. [Online]. Available: https://www.amnesty.org/en/latest/news/2017/03/fighting-back-against-cyber-censorship/.

[3] human rights watch, 2017. [Online]. Available: https://www.hrw.org/legacy/wr2k/Issues-04.htm.

[4] S. Park, "The united nations human rights council's resolution on protection of freedom of expression on the internet as a first step in protecting human rights online," *NCJ Int'l L. & Com. Reg.*, vol. 38, p. 1129, 2012.

[5] Admin, *Why streaming services block vpns amp; how to bypass vpn blocks*, Jun. 2018. [Online]. Available: https://blog.flashrouters.com/2018/09/07/why-streaming-services-block-vpns/.

[6] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981, ISSN: 00010782. DOI: 10.1145/358549.358563. [Online]. Available: http://portal.acm.org/citation.cfm?doid=358549.358563.

[7] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type III anonymous remailer protocol," *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2003-January, pp. 2–15, 2003, ISSN: 10816011. DOI: 10.1109/SECPRI.2003.1199323.

[8] *The tor project: Privacy and freedom online*, 2006. [Online]. Available: https://www.torproject.org/.

[9] *Welcome to tor metrics!* [Online]. Available: https://metrics.torproject.org/.

[10] P. F. Syverson, D. M. Goldschlag, and M. G. Reed, "Anonymous connections and onion routing," in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*, IEEE, 1997, pp. 44–54.

[11] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *International workshop on information hiding*, Springer, 1996, pp. 137–150.

[12] K. Loesing, S. J. Murdoch, and R. Dingledine, "A case study on measuring statistical data in the tor anonymity network," in *International Conference on Financial Cryptography and Data Security*, Springer, 2010, pp. 203–215.

[13] R. Dingledine and S. J. Murdoch, "Performance improvements on tor or, why tor is slow and what we're going to do about it," *Online: http://www. torproject. org/press/presskit/2009-03-11-performance. pdf*, 2009.

[14] R. Dingledine and N. Mathewson, "Anonymity loves company: Usability and the network effect.," in *WEIS*, 2006.

[15] M. Alsabah and I. Goldberg, "Performance and Security Improvements for Tor : A Survey," *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, pp. 1–41, 2014.

[16] R. Jansen, M. Traudt, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "Kist: Kernel-informed socket transport for tor," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 1, p. 3, 2018.

[17] M. Alsabah and I. Goldberg, "Performance and Security Improvements for Tor," *ACM Computing Surveys*, vol. 49, no. 2, pp. 1–36, 2016, ISSN: 03600300. DOI: 10.1145/2946802. arXiv: arXiv:1502.07526v1. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2966278.2946802.

[18] M. Perry, *The case for tor-over-quic*, Mar. 2018. [Online]. Available: https://lists.torproject.org/pipermail/tor-dev/2018-March/013026.html.

[19] J. Iyengar and M. Thomson, "Quic: A udp-based multiplexed and secure transport," *draft-ietf-quic-transport-01 (work in progress)*, 2017.

[20] R. Jansen and A. Johnson, "Safely measuring tor," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1553–1567.

[21] A. Chaabane, P. Manils, and M. A. Kaafar, "Digging into anonymous traffic: A deep analysis of the tor anonymizing network," in *2010 fourth international conference on network and system security*, IEEE, 2010, pp. 167–174.

[22] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker, "Shining light in dark places: Understanding the tor network," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5134 LNCS, pp. 63–76, 2008, ISSN: 03029743. DOI: 10.1007/978-3-540-70630-4_5.

[23] N. Hopper, "Protecting tor from botnet abuse in the long term," *Tech. rep., Tech. Rep. 2013–11-001, The Tor Project*, 2013.

[24] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[25] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM transactions on information and system security (TISSEC)*, vol. 1, no. 1, pp. 66–92, 1998.

[26] P. Boucher, A. Shostack, and I. Goldberg, *Freedom systems 2.0 architecture*, 2000.

[27] R. Dingledine, N. Mathewson, S. Murdoch, and P. Syverson, "Tor: The Second-Generation Onion Router (2014 DRAFT v1)," *Cl.Cam.Ac.Uk*, 2014. [Online]. Available: `http://www.cl.cam.ac.uk/%7B~%7Dsjm217/papers/tor14design.pdf`.

[28] C. Gulcu and G. Tsudik, "Mixing e-mail with babel," in *Proceedings of Internet Society Symposium on Network and Distributed Systems Security*, IEEE, 1996, pp. 2–16.

[29] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type iii anonymous remailer protocol," in *2003 Symposium on Security and Privacy, 2003.*, IEEE, 2003, pp. 2–15.

[30] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of tor," in *2005 IEEE Symposium on Security and Privacy (S&P'05)*, IEEE, 2005, pp. 183–195.

[31] P. Syverson and M. Reed, "Towards an Analysis of Onion Routing Security," *Designing Privacy Enhancing Technologies*, pp. 96–114, 2001.

[32] M. Perry, *Experimental Defense for Website Traffic Fingerprinting*, 2011. [Online]. Available: https:%7B%5C%%7D0A//blog.torproject.org/blog/experimental-%20defense-website-traffic-fingerprinting (visited on 2019).

[33] R. Johnson, "Touching from a Distance : Website Fingerprinting Attacks and Defenses," *CCS '12 Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 605–616, 2012.

[34] M. F. Nowlan, N. Tiwari, J. Iyengar, S. O. Amin, and B. Ford, "Fitting square pegs through round pipes: Unordered delivery wire-compatible with {tcp} and {tls}," in *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, 2012, pp. 383–398.

[35] E. Kohler, M. Handley, and S. Floyd, "Designing dccp: Congestion control without reliability," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 36, 2006, pp. 27–38.

[36] A. Langley, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, A. Riddoch, W.-T. Chang, Z. Shi, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, and I. Swett, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," *Proceedings of the Conference of the ACM Special Interest Group on Data Communication - SIGCOMM '17*, pp. 183–196, 2017, ISSN: 1552-4469. DOI: 10.1145/3098822.3098842. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3098822.3098842.

[37] T. Jager, “On the Security of TLS 1 . 3 and QUIC Against Weaknesses in PKCS # 1 v1 . 5 Encryption,”

[38] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[39] a. G. Barto, a. G. Barto, R. S. Sutton, R. S. Sutton, C. J. C. H. Watkins, and C. J. C. H. Watkins, “Sequential decision problems and neural networks,” *Advances in Neural Information Processing Systems, 2*, pp. 686–693, 1990.

[40] T. Degris, P. M. Pilarski, and R. S. Sutton, “Model-free reinforcement learning with continuous action in practice,” in *2012 American Control Conference (ACC)*, IEEE, 2012, pp. 2177–2182.

[41] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” 2014.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[43] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016. arXiv: 1509.02971.

[44] M. Liberatore, *Proposals - torspec - tor’s protocol specifications*, Feb. 2006. [Online]. Available: https://gitweb.torproject.org/torspec.git/tree/proposals/100-tor-spec-udp.txt.

[45] J. Reardon and I. Goldberg, “Improving tor using a tcp-over-dtls tunnel,” in *Proceedings of the 18th conference on USENIX security symposium*, USENIX Association, 2009, pp. 119–134.

[46] C. Viecco, “Udp-or: A fair onion transport design,” *Proceedings of Hot Topics in Privacy Enhancing Technologies (HOTPETS’08)*, 2008.

[47] M. F. Nowlan, D. I. Wolinsky, and B. Ford, “Reducing latency in tor circuits with unordered delivery,” in *Presented as part of the 3rd {USENIX} Workshop on Free and Open Communications on the Internet*, 2013.

[48] K. Loesing, S. J. Murdoch, and R. Jansen, “Evaluation of a libutp-based Tor Datagram Implementation,” pp. 1–11, 2013.

[49] M. AlSabah and I. Goldberg, “PCTCP: per-circuit TCP-over-IPsec transport for anonymous communication overlay networks,” *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS ’13*, pp. 349–360, 2013, ISSN: 15437221. DOI: 10.1145/2508859.2516715. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2508859.2516715.

[50] D. Gopal and N. Heninger, “Torchestra: Reducing interactive traffic delays over tor,” in *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*, ACM, 2012, pp. 31–42.

[51] C. Tang and I. Goldberg, “An Improved Algorithm for Tor Circuit Scheduling Categories and Subject Descriptors,” *Proceedings of the 17th ACM Conference onComputer and Communications Security*, pp. 329–339, 2010.

[52] R. Jansen and N. Hooper, "Shadow: Running tor in a box for accurate and efficient experimentation," MINNESOTA UNIV MINNEAPOLIS DEPT OF COMPUTER SCIENCE and ENGINEERING, Tech. Rep., 2011.

[53] R. Jansen and A. Johnson, "Safely Measuring Tor," 2010.

[54] F. Tschorsch and B. Scheuermann, "Tor is unfair—and what to do about it," in *2011 IEEE 36th Conference on Local Computer Networks*, IEEE, 2011, pp. 432–440.

[55] F. Tschorsch, B. Scheuermann, I. Nsdi, and F. Tschorsch, "Mind the Gap : Towards a Backpressure-Based Transport Protocol for the Tor Network This paper is included in the Proceedings of the," 2016.

[56] L. Yang and F. Li, "MTor: A multipath Tor routing beyond bandwidth throttling," *2015 IEEE Conference on Communications and NetworkSecurity, CNS 2015*, pp. 479–487, 2015. DOI: 10.1109/CNS.2015.7346860.

[57] W. B. Moore, C. Wacek, and M. Sherr, "Exploring the potential benefits of expanded rate limiting in tor: Slow and steady wins the race with tortoise," in *Proceedings of the 27th Annual Computer Security Applications Conference*, 2011, pp. 207–216.

[58] M. Alsabah, K. Bauer, and I. Goldberg, "Enhancing Tor ' s Performance using Real-time Traffic Classification Categories and Subject Descriptors," *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 73–84, 2012. DOI: 10.1145/2382196.2382208.

[59] M. AlSabah, K. Bauer, T. Elahi, and I. Goldberg, "The path less travelled: Overcoming tor's bottlenecks with traffic splitting," in *International Symposium on Privacy Enhancing Technologies Symposium*, Springer, 2013, pp. 143–163.

[60] M. Akhoondi, C. Yu, and H. V. Madhyastha, "Lastor: A low-latency as-aware tor client," in *2012 IEEE Symposium on Security and Privacy*, IEEE, 2012, pp. 476–490.

[61] A. Barton, M. Wright, J. Ming, and M. Imani, "Towards predicting efficient and anonymous tor circuits," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 429–444.

[62] T. Wang, K. Bauer, C. Forero, and I. Goldberg, "Congestion-aware path selection for tor," in *International Conference on Financial Cryptography and Data Security*, Springer, 2012, pp. 98–113.

[63] S. Chinchali, P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone, and S. Katti, "Cellular network traffic scheduling with deep reinforcement learning.," in *AAAI*, 2018, pp. 766–774.

[64] S. M. Jafari, M. Taghipour, and M. Meybodi, "Bandwidth allocation in wimax networks using reinforcement learning," *World Applied Sciences Journal*, vol. 15, no. 4, pp. 525–531, 2011.

[65] H. Xu, Y. Zu, F. Shen, F. Yan, F. Qin, and L. Shen, "Fair resource allocation based on deep reinforcement learning in fog networks," in *International Conference on Ad Hoc Networks*, Springer, 2019, pp. 135–148.

[66] Y. Wei, F. R. Yu, M. Song, and Z. Han, "User scheduling and resource allocation in hetnets with hybrid energy supply: An actor-critic reinforcement learning

approach," *IEEE Transactions on Wireless Communications*, vol. 17, no. 1, pp. 680–692, 2017.

[67] B. Yan, Y. Zhao, Y. Li, X. Yu, J. Zhang, Y. Wang, L. Yan, and S. Rahman, "Actor-critic-based resource allocation for multi-modal optical networks," in *2018 IEEE Globecom Workshops (GC Wkshps)*, IEEE, 2018, pp. 1–6.

[68] J. Iyengar and M. Thomson, "Quic: A udp-based multiplexed and secure transport; draft-ietf-quic-transport-24," *Internet Engineering Task Force: Newark, DE, USA*, 2019.

[69] M. Perry, *Tor's open research topics: 2018 edition*, Jul. 2018. [Online]. Available: `https://blog.torproject.org/tors-open-research-topics-2018-edition`.

[70] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How secure and quick is quic? provable security and performance analyses," in *2015 IEEE Symposium on Security and Privacy*, IEEE, 2015, pp. 214–231.

[71] M. Fischlin and F. Günther, "Multi-stage key exchange and the case of google's quic protocol," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1193–1204.

[72] J. Geddes, R. Jansen, and N. Hopper, "IMUX: Managing Tor Connections from Two to Infinity, and Beyond," *Proceedings of the 13th Workshop on Privacy in the Electronic Society - WPES '14*, pp. 181–190, 2014, ISSN: 15437221. DOI: `10.1145/2665943.2665948`. arXiv: `arXiv:1010.2697v1`. [Online]. Available: `http://dl.acm.org/citation.cfm?doid=2665943.2665948`.

[73] Sandvine Incorporated ULC, "The Global Internet Phenomena Report: october 2018," no. October, p. 4, 2018. DOI: 10.1117/12.890392. [Online]. Available: %7Bhttps://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf%7D.

[74] N. Unger, *NetMirage*, 2017. [Online]. Available: https://crysp.uwaterloo.ca/software/netmirage/.

[75] R. Jansen, K. S. Bauer, N. Hopper, and R. Dingledine, "Methodically modeling the tor network.," in *CSET*, 2012.

[76] R. Jansen, P. Syverson, and N. Hopper, "Throttling Tor Bandwidth Parasites," *2012 USENIX Security Symposium*, pp. 349–364, 2012.

[77] R. Jansen and M. Traudt, "Tor's Been KIST: A Case Study of Transitioning Tor Research to Practice," 2017. arXiv: 1709.01044. [Online]. Available: http://arxiv.org/abs/1709.01044.

[78] D. Gopal and N. Heninger, "Torchestra: Reducing Interactive Traffic Delays over Tor," *Proceedings ofthe 2012 ACM Workshop on Privacy in the Electronic Society (WPES 2012), pages 31–42. ACM, 2012*, 2012.

[79] Crow, MachMetrics, and StapleCactus, *Website size: The average web page size is more than 2mb – twice the size of the average page just 3 years ago*, Jul. 2018. [Online]. Available: https://www.machmetrics.com/speed-blog/website-size-the-average-web-page-size-is-more-than-2mb-twice-the-size-of-the-average-page-just-3-years-ago/.

[80] R. Jansen, M. Traudt, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "KISt: Kernel-informed socket transport for ToR," *ACM Transactions on Privacy and*

Security, vol. 22, no. 1, pp. 1–37, 2018, ISSN: 24712574. DOI: 10.1145/3278121.

[81] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the impact of video quality on user engagement," in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 41, 2011, pp. 362–373.

[82] A. Kwon, M. Alsabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit Fingerprinting Attacks : Passive Deanonymization of Tor Hidden Services," *USENIX Security*, pp. 287–302, 2015.

[83] K. Bauer, D. Mccoy, and D. Grunwald, "Low-Resource Routing Attacks Against Tor," *Proc. of Workshop on Privacy in Electronic Society*, 2007.

[84] P. Mayank, "Tor Traffic Identification," *7th International Conference on Communication Systems and Network Technologies*, pp. 85–91, 2017. DOI: 10.1109/CSNT.2017.17.

[85] G. He, M. Yang, X. Gu, and J. Luo, "A Novel Active Website Fingerprinting Attack against Tor Anonymous System," *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 112–117, 2014. DOI: 10.1109/CSCWD.2014.6846826.

[86] D. Herrmann, R. Wendolsky, and H. Federrath, "Website Fingerprinting : Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier," *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 31–41, 2009.

[87] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website Fingerprinting in Onion Routing Based Anonymization Networks Categories and Subject Descriptors," *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pp. 103–113, 2011.

[88] M. Perry, *Experimental Defense for Website Traffic Fingerprinting*, 2011. [Online]. Available: https://blog.torproject.org/experimental-defense-website-traffic-fingerprinting.

[89] M. Edman and P. Syverson, "AS-awareness in Tor Path Selection," *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 380–389, 2009.

[90] N. Hopper, E. Y. Vasserman, and E. Chan-tin, "How Much Anonymity does Network Latency Leak ?" *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 2, 2010. DOI: 10.1145/1698750.1698753.

[91] N. Mathewson and M. Perry, "Towards Side Channel Analysis of Datagram Tor vs Current Tor," pp. 1–9, 2018.

[92] J. Geddes, R. Jansen, and N. Hopper, "How low can you go: Balancing performance with anonymity in Tor," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7981 LNCS, pp. 164–184, 2013, ISSN: 03029743. DOI: 10.1007/978-3-642-39077-7_9.

[93] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting,"

in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 215–226.

[94] D. Claudia, S. Seys, J. Claessens, B. Preneel, and K. U. L. Esat-cosic, "Towards measuring anonymity," *PET'02 Proceedings of the 2nd international conference on Privacy enhancing technologies*, pp. 54–68, 2002.

[95] A. Iacovazzi, S. Sarda, and Y. Elovici, "I NFLOW : Inverse Network Flow Watermarking for Detecting Hidden Servers," *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 747–755, 2018.

[96] X. Wang and S. Chen, "Network Flow Watermarking Attack on Low-Latency Anonymous Communication Systems," *IEEE Symposium on Security and Privacy (SP '07)*, 2007.

[97] A. Serjantov and G. Danezis, "Towards an information theoretic metric for anonymity," in *International Workshop on Privacy Enhancing Technologies*, Springer, 2002, pp. 41–53.

[98] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[99] C. Wengerter, J. Ohlhorst, and A. G. E. von Elbwart, "Fairness and throughput analysis for generalized proportional fair frequency scheduling in ofdma," in *2005 IEEE 61st vehicular technology conference*, IEEE, vol. 3, 2005, pp. 1903–1907.

[100] F. P. Kelly, A. K. Maulloo, D. K. H. Tan, F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks : shadow prices , proportional fairness and stability Rate control for communication networks

: shadow prices , proportional fairness and stability," vol. 5682, 2017. DOI: 10.1057/palgrave.jors.2600523.

[101] S. W. Cho and A. Goel, "Bandwidth allocation in networks: A single dual update subroutine for multiple objectives," *Lecture Notes in Computer Science*, vol. 3405, pp. 28–41, 2005, ISSN: 03029743. DOI: 10.1007/11527954_4.

[102] L. Li, M. Pal, and Y. R. Yang, "Proportional fairness in multi-rate wireless lans," in *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, IEEE, 2008, pp. 1004–1012.

[103] *State of the web*. [Online]. Available: https://httparchive.org/reports/state-of-the-web.

[104] R. K. Jain, D.-M. W. Chiu, W. R. Hawe, *et al.*, "A quantitative measure of fairness and discrimination," *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, 1984.