NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

HUGO GAMALIEL DOS SANTOS PEREIRA

BSc Degree in Computer Science and Engineering

MIRACE: MULTI-PATH INTEGRATED ROUTING ARCHITECTURE FOR CENSORSHIP EVASION

PRIVACY-PRESERVED INTERNET COMMUNICATION WITH UNOBSERVABLE AND ANONYMOUS MIXNET TRAFFIC

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
September, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

MIRACE: MULTI-PATH INTEGRATED ROUTING ARCHITECTURE FOR CENSORSHIP EVASION

PRIVACY-PRESERVED INTERNET COMMUNICATION WITH UNOBSERVABLE AND ANONYMOUS MIXNET TRAFFIC

HUGO GAMALIEL DOS SANTOS PEREIRA

BSc Degree in Computer Science and Engineering

Adviser: Henrique João Lopes Domingos
Associate Professor, NOVA University Lisbon

Examination Committee

Chair: Carlos Augusto Isaac Piló Viegas Damásio
Associate Professor, NOVA University Lisbon

Members: Diogo Miguel Barrinha Barradas
Assistant Professor, University of Waterloo

Henrique João Lopes Domingos
Associate Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
September, 2024

MIRACE: Multi-Path Integrated Routing Architecture for Censorship Evasion
Privacy-Preserved Internet Communication with Unobservable and Anonymous Mixnet Traffic

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.5) [75].

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Professor Henrique Domingos, for his exceptional guidance, support, and invaluable feedback throughout this journey. His expertise and encouragement have been instrumental in completing this thesis.

I am also sincerely thankful to Afonso Vilalonga for his constant support and thoughtful discussions. His readiness to assist and offer feedback during critical moments was invaluable to developing my work.

My gratitude extends to Professor João Lourenço for his guidance during the Undergraduate Research Opportunity Program, which ignited my passion for research and laid the foundation for my academic development and critical thinking.

On a personal note, I want to thank my girlfriend, Beatriz Chaveiro, whose encouragement and constant support have been a source of strength and motivation throughout this process.

I also want to express my heartfelt appreciation to my friends for their companionship and sharing joyful moments.

Lastly, my deepest thanks go to my parents, Vitor Pereira and Isabel Pereira, for their endless love, support, and belief in me. Their encouragement and sacrifices have been my most important source of strength, inspiring me to face challenges with resilience and strive to do my best every step along the way.

Abstract

Contemporary research has underscored the alarming surveillance and censorship practices of totalitarian regimes and government agencies in observing traffic in communication networks, including the Internet. In response to these pressing challenges, anonymization networks have gained prominence in the digital landscape. Among these, Tor stands out as one of the most popular solutions, playing an essential role in protecting user privacy and anonymity, combating online censorship, and upholding the fundamental rights of free expression and communications. However, recent studies have uncovered vulnerabilities in the Tor network, including risks of deanonymization, often exploited with fingerprinting or correlation attacks launched by state-level adversaries or through the collaboration of multiple ones.

In this dissertation, we examine the issues of anonymity and privacy breaches within anonymization networks. To address these concerns, we present MIRACE - a multipath integrated routing architecture for a communication system that enhances privacy through mixed circuits, making communications resilient to tracing, blocking, and thus censorship. MIRACE mixed circuits are established through covert channels layered upon TLS, QUIC tunnels and WebRTC media streams, with different segments of a single circuit potentially using distinct covert methods. Our architecture allows traffic to be split across N circuits composed of M nodes, and each node incorporates advanced techniques like traffic encapsulation, shaping, and induced jitter. To the extent of our knowledge, our solution emerges as a pioneering contribution by combining per-packet traffic splitting with multi-protocol encapsulation, and it aims at bolstering the defences against state-of-the-art Internet censorship mechanisms. Moreover, the techniques addressed in this proposal are also potential candidates for future enhancements of Tor.

We have implemented a prototype and performed extensive validations to observe the correctness and experimental performance of MIRACE. The obtained results show that the proposed solution operates as expected with throughput on the order of Mbps and maintains latency conditions suitable for diverse application contexts and usage scenarios. Furthermore, the proposed system is also resistant to unobservability evaluations regarding website fingerprinting.

Keywords: Anonymization Communication Systems, Privacy-Preserving Communication, Internet Censorship, Anti-censorship, Mixed Circuits, Traffic Randomization and Shaping, Traffic Splitting, Traffic Encapsulation, Website Fingerprinting, Traffic Correlation Attacks

Resumo

A investigação contemporânea sublinhou as alarmantes práticas de vigilância e censura dos regimes totalitários e das agências governamentais na observação do tráfego nas redes de comunicação, incluindo a Internet. Em resposta a estes desafios prementes, as redes de anonimato ganharam destaque no panorama digital. Entre estas, o Tor destaca-se como uma das soluções mais populares, desempenhando um papel essencial na proteção da privacidade e do anonimato dos utilizadores, no combate à censura online e na defesa dos direitos fundamentais de liberdade de expressão e comunicação. No entanto, estudos recentes revelaram vulnerabilidades na rede Tor, incluindo riscos de desanonimização, muitas vezes explorados com impressões digitais de websites ou ataques de correlação lançados por adversários a nível estatal ou através da colaboração de múltiplos adversários.

Nesta dissertação, examinamos as questões do anonimato e das violações de privacidade nas redes de anonimato. Para abordar estas preocupações, apresentamos o MIRACE - uma arquitetura de encaminhamento integrado multipercurso para um sistema de comunicação que aumenta a privacidade através de circuitos mistos, tornando as comunicações resilientes ao rastreio, bloqueio e, portanto, à censura. Os circuitos mistos MIRACE são estabelecidos através de canais secretos dispostos em camadas sobre TLS, túneis QUIC e fluxos de media WebRTC, com diferentes segmentos de um único circuito a utilizarem potencialmente métodos secretos distintos. A nossa arquitetura permite que o tráfego seja dividido em N circuitos compostos por M nós, e cada nó incorpora técnicas avançadas como encapsulamento de tráfego, modelação e jitter induzido. Tanto quanto sabemos, a nossa solução surge como uma contribuição pioneira ao combinar a divisão de tráfego por pacote com encapsulamento em múltiplos protocolos, e visa reforçar as defesas contra os mecanismos de censura da Internet de última geração. Além disso, as técnicas abordadas nesta proposta são também potenciais candidatas para futuras melhorias do Tor.

Implementámos um protótipo e realizámos extensas validações para observar a correção e o desempenho experimental do MIRACE. Os resultados obtidos mostram que a solução proposta opera como esperado, com um throughput na ordem dos Mbps e mantém condições de latência adequadas para vários contextos de aplicação e cenários de utilização. Além disso, o sistema proposto é também resistente a avaliações de

inobservabilidade em relação à impressão digital de websites.

Palavras-chave: Sistemas de Comunicação com Anonimato, Privacidade na Comunicação, Censura na Internet, Anticensura, Randomização e modelação de tráfego, Divisão de tráfego, Encapsulamento de tráfego, Impressão digital do site, Ataques de correlação

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Context and Scope

The exchange of information over the Internet has become an essential aspect of modern society. However, with this growing popularity, Internet censorship and surveillance have also witnessed unprecedented growth, posing a significant threat to online privacy and free expression [41]. Totalitarian states, such as China [37, 127], Turkmenistan [87], Russia [95], and Iran [77], have been particularly interested in monitoring and censoring Internet communications. These regimes deploy sophisticated apparatuses to control and suppress information, driven by diverse motivations such as political control, economic interests, and security concerns [86]. Methods of censorship include blocking communications through DNS manipulation [86], blocking specific protocols [36], keyword filtering [24], and monitoring online platforms [63].

The growing prevalence of censorship has created significant obstacles to maintaining privacy in online communications, especially with the fast advancement of technology and the increasing exchange of sensitive information [126]. In response to these escalating challenges to online freedom, anonymization networks have emerged as a popular solution for individuals seeking to safeguard the privacy of their communications [17, 20, 32, 69, 93, 113, 122, 131]. These networks strive to obfuscate the linkage between the participating entities with what they are accessing, enabling users to communicate and access information without fear of censorship or surveillance [16].

Tor [32], short for "The Onion Router", is a well-known anonymity network that uses onion routing to shield user identities and online activities from prying eyes. As a low-latency anonymous network [84], it is suitable for applications such as web browsing. In this system, messages undergo a multi-layered encryption process before traversing a circuit of interconnected nodes. These circuits are dynamically generated and typically contain three Tor nodes: an entry, a middle, and an exit node. At each node, only the outermost layer of encryption is decrypted, ensuring that each node knows its predecessor and successor and, as such, none of the nodes simultaneously know both the sender and receiver.

Tor's commitment to anonymity has made it a powerful tool for protecting free speech, making it an ideal environment for whistleblowing, journalistic activities, and citizens in oppressive regimes [76]. However, censorship authorities continuously develop new monitoring techniques, which can be used to undermine user privacy and potentially compromise the anonymity provided by Tor, and they are usually up to date on current research advancements. Recent studies have revealed vulnerabilities within the Tor network, particularly through website fingerprinting [19] and correlation attacks launched by global adversaries or via collaboration between multiple adversaries [46, 73, 80, 85, 89, 96, 109].

Various strategies have been employed to combat these vulnerabilities, such as pluggable transport systems, initially designed to bypass blocking attempts of the Tor protocol and entry nodes by disguising or randomizing Tor traffic. However, some pluggable transports may also be helpful against traffic correlation attacks [114]. Despite these efforts, the constant arms race between censorship techniques and privacy measures has rendered some solutions obsolete or ineffective against advanced censorship methods [51, 119].

1.2 Motivation

The evolving landscape of online surveillance demands further enhancements to privacy mechanisms, particularly as the Tor Network faces increasingly intense scrutiny. While improving privacy can come at the cost of performance [25], the popularity of Tor makes it essential to reinforce its defense mechanisms in response to the persistent efforts of powerful adversaries. Governments [94], law enforcement agencies [60], and other adversaries continue to seek methods to compromise user anonymity within the Tor Network, highlighting the need for innovative solutions that preserve Internet users' anonymity while maintaining low latency.

One significant limitation of Tor is its vulnerability to deanonymization attacks due to a lack of traffic characteristic obfuscation. These attacks underscore the pressing need for more dynamic solutions, such as traffic obfuscation, randomization, shaping, encapsulation, and splitting. Traffic obfuscation and shaping involve altering the censorship evasion system's traffic to make it less distinguishable to external observers [34]. Traffic encapsulation involves wrapping the user's communication within another protocol, such as WebRTC, QUIC, or TLS, making the traffic appear to be part of a communication associated with the carrier protocol rather than the original one, helping to obfuscate the true nature of the traffic [11, 40, 114]. Finally, traffic splitting divides a traffic flow across multiple network paths, making it harder for an adversary to capture the complete traffic flow of a user [50, 92, 120].

Some techniques can more easily be incorporated into the Tor network than others. For example, Tor's reliance on TCP rather than UDP and its flow-based approach introduces intrinsic limitations in providing traffic splitting at the packet level, as individual packets cannot be easily routed through distinct paths, resulting in a fixed path per session.

Additionally, Tor's consensus-driven development process can make it difficult to adapt quickly to evolving censorship methods and privacy needs. These constraints highlight the need for a dynamic solution that performs reasonably for everyday browsing and video streaming while allowing for tailored, more privacy-enhancing configurations.

1.3 Proposed Solution and Goals

To address these challenges, this thesis proposes an Internet censorship circumvention system, MIRACE (Multi-Path Integrated Routing Architecture for Censorship Evasion), that provides secure transmission of Internet communications through dynamically constructed, mixed circuits. This approach is designed to make communications resilient to tracing, blocking, and censorship. Our architecture allows traffic to be split across a user-configured number of circuits, each composed of multiple nodes and employs multi-layered encryption strategies. The dynamic nature of traffic splitting and circuit selection, determined by heuristic methods like packet count or time intervals and round-robin or weights-based selection, strengthens the system's resistance to sophisticated correlation attacks and traffic analysis strategies, including those using machine learning and deep learning techniques and at the same time, maintaining decent throughput and latency conditions for regular online activities such as web browsing.

Our solution also incorporates traffic encapsulation, allowing users to choose the encapsulation type for their network paths. Options include encapsulating traffic in video frames from WebRTC video conferences or within QUIC or TLS connections. A key aspect of our approach, to the best of our knowledge not implemented by any other system, is that we allow circuits to employ different encapsulation types across various segments of the same circuit. This WebRTC-based encapsulation methodology draws inspiration from existing literature, particularly TorKameleon [114], previously developed at NOVA LINCS. This pluggable transport system relies on WebRTC-based tunnels to establish covert communications and uses K-Anonymization techniques to fragment and reroute traffic via multiple paths. This approach ensures that if the censor is not monitoring all the proxies that make up the network, it will only see a portion of the users' traffic while also making the traffic between different users appear identical, giving each user plausible deniability of his traffic. However, unlike TorKameleon, which fixes circuits for each stream or HTTP request, our solution enables various packets of the same stream or request to be transmitted via distinct paths. Additionally, TorKameleon's traffic does not undergo the same multilayered encryption as our solution. In TorKameleon, the traffic is only encrypted between each pair of communicating proxies, meaning any intermediary proxy can still see both the origin and the destination of the traffic.

Therefore, the **main goal** of this thesis was to create a dynamic communication environment capable of withstanding state-of-the-art website fingerprinting attacks while maintaining decent throughput and latency conditions for practical use in different application contexts. To achieve this goal, we developed the network structure from scratch and

incorporated advanced techniques like packet-level traffic splitting, traffic encapsulation, and traffic shaping, building upon the principles of Onion Routing to ensure anonymity and privacy with end-to-end guarantees.

1.4 Contributions

The contributions of this work can be summarised as follows:

- Design of a Privacy-Preserved Internet Communication Solution with traffic splitting (packet-level), shaping, and encapsulation through WebRTC-based covert channels and secure TLS and QUIC tunnelling, where each segment of the circuit might be encapsulated with a different type from our supported options.
- An open-source prototype implementation of the proposed solution, making it accessible to researchers and practitioners for studies and possible future improvements.
- A comprehensive experimental evaluation of our solution encompassing an assessment of usability, system resource utilization, and performance trade-offs. Additionally, we performed experiments to evaluate the system's resilience specifically against website fingerprinting attacks.

1.5 Report Organization

The remainder of this dissertation is structured as follows: First, Chapter 2 comprehensively provides an overview of anonymization networks, clarifying their fundamental principles and mechanisms. This chapter delves into exploring different traffic analysis methodologies and offers a detailed explanation of multipath strategies and how they can be used to achieve better performance and anonymity. Chapter 3 serves as the foundation of this dissertation and includes relevant references covering various aspects of related work aligned with the goals and expected contributions. Next, Chapter 4 presents the specifications of the proposed solution, more precisely, the system model, the architecture of the system and the adversary model. This sets the stage for the next chapter, Chapter 5, which describes the implementation details of the proposed solution, leading into the experimental evaluation chapter, Chapter 6. Lastly, Chapter 7 presents the concluding remarks of this thesis and offers insight into future work.

2

Background

This chapter delves into the essential background knowledge necessary for understanding the context and challenges this dissertation addresses. Within Section 2.1, we explore the concept of Anonymization Networks, its different architectures, and inherent constraints. In Section 2.2, we investigate various traffic analysis methodologies that threaten the anonymity and privacy of Internet users. Lastly, Section 2.3 presents multipath strategies as a critical network resilience and anonymity enhancement component.

2.1 Anonymization Networks

Anonymization networks are critical in maintaining online privacy and protecting users from surveillance and censorship. These systems decouple users from the destinations they access, ensuring that external observers cannot simultaneously deduce the sender and receiver of traffic flows. Anonymization networks are typically classified into two categories: high-latency and low-latency networks. High-latency networks prioritize anonymity by introducing significant delays, making them suitable for applications where privacy is more critical than speed. In contrast, low-latency networks minimize delays, offering faster performance but potentially reduced anonymity (e.g., vulnerable to correlation attacks). Although a third concept, the concept of medium latency networks, is mentioned in the literature, it remains loosely defined and is still under exploration [25, 116].

2.1.1 High Latency Systems

High-latency anonymity systems [22, 48, 81] are designed to provide the strongest possible protection of user anonymity by introducing substantial delays in data transmission. Unlike real-time communication systems, these typically follow a message-based model, where data is transmitted in batches rather than instantly. As a result, messages may take several hours to days to reach their destination, making these systems impractical for scenarios requiring immediate responses. However, they are ideally suited for situations where the privacy and security of the communication are paramount.

High-latency systems employ sophisticated techniques such as sending false packets and reordering messages before transmission to further obscure the relationship between senders and receivers. These methods make it highly challenging for adversaries to analyze traffic flows and correlate incoming and outgoing messages. By creating complex and unpredictable traffic patterns, these systems ensure a high degree of anonymity but impose limitations on their use in everyday applications where real-time communication is needed.

Example: Mixminion

Mixminion [22] is a well-known high-latency anonymous remailer system. It employs a technique where messages are sent through multiple intermediate nodes, each introducing deliberate delays and removing a layer of encryption, thereby obscuring the relationship between the sender and the recipient. One of the key innovations of Mixminion is message splitting, which further strengthens anonymity by dividing each message into smaller chunks and sending these parts along different paths through the network. This approach ensures that even if attackers compromise some nodes, they cannot reconstruct the entire message.

Additionally, Mixminion supports single-use reply blocks (SURBs), which allow recipients to send anonymous replies without exposing their identity. Unlike traditional systems, where sending a response could reveal the communication path, as replies may be rare relative to forward messages, SURBs are designed to enable secure and unlinkable replies, further complicating any efforts by an adversary to correlate messages. As such, the nodes cannot distinguish them, so forward and reply messages share the same anonymity set.

Meanwhile, Mixminion provides strong privacy guarantees to defeat even a global passive adversary. Splitting messages, routing them along different paths, and using SURBs for anonymous replies ensures that no single node or adversary can trace the communication. These techniques and added latency to prevent timing analysis make Mixminion highly resilient against attempts to link senders and recipients.

2.1.2 Low Latency Systems

Low latency anonymity systems [17, 32, 131] offer the lowest response times and are among the most widely used anonymity solutions. They are optimized to deliver fast data transfer and minimal delays, making them ideal for real-time applications such as web browsing, instant messaging, and Voice over IP (VoIP). However, this speed comes at a cost: low-latency systems provide less privacy than higher-latency alternatives [101]. These systems are more vulnerable to traffic correlation and timing attacks, where adversaries can analyze patterns in network traffic to deanonymize users or potentially infer the content they are accessing.

Low latency systems, such as Tor [32], are typically circuit-based. They operate by routing data through a sequence of intermediary nodes, which encrypt and decrypt traffic to obscure its origin and destination. While this provides a basic level of anonymity, these systems generally lack specific protections against correlation attacks. As a result, adversaries may still be able to trace traffic flow from sender to receiver. Despite these limitations, low latency systems remain popular because they allow users to perform more complex, latency-sensitive tasks that most internet users consider essential. For instance, tasks like loading web pages quickly or participating in live communication are only feasible with such systems.

The anonymity guarantees of low latency systems are typically evaluated through experimental studies, which assess their resilience against potential attacks under real-world conditions [116]. These evaluations help to quantify the level of privacy protection that users can expect, allowing system developers to refine these solutions and address emerging threats. Despite their inherent vulnerabilities, low latency systems like Tor are critical in providing accessible anonymity for everyday users who require privacy and usability in their online interactions.

Example: Tor Network

Tor [32], short for "The Onion Router", is one of the most renowned anonymization networks, boasting a user base exceeding 2 million individuals per day and an extensive infrastructure comprising thousands of publicly accessible routers[1]. This large user base helps provide a layer of anonymity by having users "blend in with the crowd". Tor is a circuit-based, anonymous, low-latency communication network rooted in Onion Routing, a distributed overlay network developed to anonymize TCP-based applications.

To build a specific path to route traffic, each user runs a Tor daemon. This daemon handles the connection with user applications and establishes circuits within the Tor Network by establishing session keys, which are symmetric keys, with the circuit nodes. After selecting the circuit, Tor's incoming traffic is encrypted with the shared keys in multiple layers, ensuring that each node can only peel off one layer of encryption while passing the data along, because it does not know the other keys. Moreover, the responses follow the same path but in reverse, with each node adding its own layer of encryption. The final user then decrypts all layers to reveal the original message.

This multi-layered design ensures that each relay knows only about its immediate predecessor and successor within the circuit. These circuits typically comprise only an entry, a middle, and an exit relay node, granting that only the entry node knows the user's IP address. In contrast, the exit relay connects to the desired destination, ensuring unlinkability between the incoming and outgoing flows. Another security feature of Tor is that it forwards Tor cells, 512 bytes each, making it overly complicated for an attacker to detect the accurate size of files transferred by separated connection streams [103].

[1]Tor Metrics - https://metrics.torproject.org (Accessed on 10 October 2023)

2.1.3 Medium Latency Systems

Medium latency anonymity systems [30, 88, 93] balance the speed of low latency systems and the stronger privacy guarantees of high latency systems. These systems aim to offer enhanced privacy protections without imposing severe delays, making them usable for many online tasks. By introducing moderate delays and additional defences, such as increased bandwidth usage for fake traffic, medium latency systems improve user anonymity while maintaining an acceptable level of performance for non-real-time applications.

Medium latency systems typically focus on message-based communication, where real-time interaction is less critical, but privacy remains a priority. They incorporate techniques to obscure the relationship between senders and receivers, often using batching, mixing, or padding mechanisms to prevent adversaries from correlating traffic flows. Although these systems share some commonalities with high latency solutions, they are optimized to reduce delay without sacrificing all privacy protections.

Efforts have been made to extend the principles of circuit-based, low-latency systems to create medium-latency systems with better defences against traffic correlation attacks [116].

Example: Loopix

Loopix [93] is an anonymity system created in 2017 that uses a message-oriented architecture based on mixed network design. Unlike circuit-oriented networks, such as Tor, which rely on fixed routes or circuit, Loopix employs a Poisson mix strategy, where messages follow randomized paths. This approach enhances user privacy and security by introducing unpredictability into the message routing process.

Despite being designed for low-latency communications, it is worth noting that some papers classify Loopix as a medium-latency system [116], despite the fact that the term 'medium-latency' does not have a concrete or universally agreed definition.

Loopix represents a significant advancement in mix network technology, making message-based systems more competitive with existing onion routing solutions that have dominated the field since Tor's inception. Loopix also provides forward and backward secrecy, ensuring that even if an adversary compromises a mix node, they cannot decrypt past or future messages.

2.2 Traffic Analysis Methodologies

The anonymity provided by the networks mentioned earlier is subject to intense scrutiny from adversaries, including employing traffic analysis techniques to unveil hidden information about the network's users and activities. Understanding the methodologies used for traffic analysis is crucial when developing robust privacy solutions to comprehend

the vulnerabilities that might compromise user privacy. This section explores various traffic analysis methodologies employed to deanonymize users.

2.2.1 Passive Correlation Attacks

One of the most well-known methodologies designed to deanonymize users within anonymity networks is Passive Correlation Attacks. As the name suggests, these attacks rely on passive observation and the analysis of network traffic to correlate and disclose the source and destination of communications. Adversaries who conduct passive correlation attacks avoid proactively interfering with the operation of the network, making it more difficult to perceive that traffic is being scrutinized. Thus, they are based on passive monitoring of traffic patterns and characteristics to make inferences and deanonymize the network's users.

Passive Correlation Attacks frequently involve sophisticated statistical analysis [78, 108], encompassing traffic flow analysis, payload inspection and timing analysis. Advanced pattern recognition techniques, exemplified by state-of-the-art tools such as MixFlow [5], FlowTracker [46], DeepCorr [85], DeepCoFFEA [89] and DeepMetricCorr [73], leverage cutting-edge machine and deep learning techniques to identify concealed patterns within traffic that may not be apparent through traditional statistical analysis.

2.2.2 Active Correlation Attacks

On the other hand, active correlation attacks go beyond just passive traffic monitoring. Adversaries in these attacks strategically inject delays into specific network segments to mark and track the flow of packets. These deliberate delays create identifiable characteristics in the traversing packets, facilitating user correlation with activities. Consequently, by marking flows with delays at the entry of the network and monitoring their behaviour upon exit, adversaries confirm suspicions and correlate users with their activities.

Despite the possibility of being more efficient in achieving traffic correlation through intensively tampering with network flows, they also carry a higher risk of detection due to higher latency in communications, which can raise alarms and trigger countermeasures. However, they require the analysis of fewer flows, and they have a lower computation overhead, as they do not need to monitor multiple communication flows for long periods of time. Some of the most well-known active correlation attacks use flow watermarking [54, 55, 71] and flow fingerprinting [96].

Flow watermarking, a notable active correlation attack technique, involves the subtle embedding of a marker, typically just a single bit, within the flow. This marker functions as a binary indicator, signalling the presence or absence of the watermark in the flow. It provides limited information, primarily aimed at confirming the existence of the embedded signal without conveying extensive details for identification purposes. Flow fingerprinting is a more sophisticated method for transmitting multiple bits of information within the flow.

This technique facilitates embedding diverse data elements, potentially encompassing unique identifiers linked to individual users or other specific markers.

Website Fingerprinting

Website Fingerprinting (WF) is highly relevant to this work, representing a key attack vector we aim to mitigate. Given its effectiveness against low-latency systems and the growing sophistication of machine and deep learning classifiers, we will evaluate our solution specifically against WF attacks to assess its resilience and privacy guarantees.

Website Fingerprinting is a specific type of traffic analysis attack that focuses on identifying the websites a user is visiting based solely on encrypted traffic patterns, such as packet sizes, timings and directions of packets. This attack is particularly effective against low-latency systems, such as Tor, where adversaries can exploit the unique characteristics of web traffic to infer the websites being accessed [49, 106, 123].

The adversary collects a dataset of traffic flows from known websites and uses machine and deep learning algorithms to train a classifier to distinguish between websites based on their traffic patterns. Several studies propose WF attacks using traditional machine learning models such as Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Random Forest (RF) [102].

In recent years, deep learning models have improved the accuracy of WF attacks. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) [21, 72] have shown promise in learning more complex and subtle patterns within the traffic data, thereby boosting the performance of classifiers.

2.2.3 Other Attacks

In addition to abovementioned attacks, several other attack methodologies pose potential threats to the privacy and anonymity of users within anonymity networks. These attacks often exhibit specialized characteristics and may capitalize on network vulnerabilities or user behaviours. Some simple but powerful attacks include Congestion and Denial of Service Attacks, where the network is flooded with traffic, disrupting its regular functioning or, in extreme cases, to complete unusability.

Other pertinent disruptions include the Sybil Attack [33], where adversaries deploy a multitude of malicious nodes within the network and create the illusion that those nodes pertain to different entities, aiming to achieve control or influence, impacting the effectiveness of multipath routing and user anonymity by ranging from traffic redirection or eavesdropping on user communications.

Moreover, adversaries may actively intercept communications between a sender and receiver through Interception Attacks, gaining access to the communication's content and metadata. This can be achieved by compromising nodes or exploiting vulnerabilities, allowing adversaries to eavesdrop on communications and collect information about the participants involved [109]. In contrast, passive correlation attacks do not involve

direct access to the communication. Instead, adversaries analyze traffic patterns, such as timing and packet size, across multiple points in the network to infer which users are communicating. These attacks rely on statistical metadata analysis rather than capturing the content, distinguishing them from interception-based methods.

Adversaries can differentiate users' traffic patterns through repeated observations (as a user may tend to visit the same destinations over time), especially if we consider that users generally access a limited collection of servers when using anonymization networks. As such, statistical disclosure attacks can capitalize on these patterns by intersecting different sets of users at any moment [90].

2.3 Multipath Strategies and Traffic Splitting

After thoroughly examining various anonymization networks and their vulnerabilities, it is essential to understand the concept of multipath strategies. These strategies are not only used for improving network performance, but they also offer significant advantages in terms of resistance to certain attacks. By transmitting data over multiple paths concurrently, traffic splitting may enhance network resilience, reliability, and resistance against various adversarial attacks.

Multipath strategies represent a network routing paradigm where data is distributed across multiple paths simultaneously. This approach, used in systems like Loopix [93], diverges from traditional single-path routing, where data is transmitted sequentially via a fixed series of relays. Distributing traffic over multiple routes may offer several advantages: enhanced fault tolerance, load balancing, and optimized resource utilization. Moreover, from a threat standpoint, multipath strategies (e.g. traffic splitting) may obscure traffic patterns and introduce irregularities in traffic flow, making it more difficult for adversaries to perform correlation attacks.

It is worth noting that the widely-used Tor network currently relies on single-path routing for each stream, multiplexing multiple streams within the same circuit. While this approach is straightforward, it does not effectively optimize resource use or mitigate congestion. Several proposals have been made to address these limitations in Tor to enhance its performance and unobservability. These proposals range from increasing network capacity [56, 83] and optimizing path selection [2] to employing multipath routing and traffic splitting [28, 50, 66, 74, 92, 97, 120, 129]. Multipath routing schemes distribute network traffic across several paths concurrently, offering both performance improvements, such as reduced latency and better throughput, and enhanced resistance to passive and active correlation attacks. This dual benefit makes multipath routing an attractive option for improving anonymous communication systems.

Due to their strengths, protocols like UDP and MPTCP are often considered when implementing multipath strategies. Being lightweight and connectionless, UDP allows for quick transmissions with minimal overhead, although it lacks built-in reliability mechanisms. MPTCP, on the other hand, enables a single connection to utilize multiple paths

concurrently, dynamically redistributing traffic to optimize performance and reliability. While Tor currently eschews these protocols in favour of TCP-based multiplexing over a single path, research has shown that using multipath approaches, even with TCP, can significantly enhance both performance and resistance to various forms of traffic analysis and deanonymization [3, 66].

These studies underscore multipath strategies' advantages in improving network performance and bolstering security against traffic analysis and website fingerprinting attacks [27, 28]. By introducing unpredictability into traffic flows and reducing the likelihood of adversaries controlling all paths simultaneously, multipath strategies make monitoring or compromising anonymous communication systems significantly more difficult. Distributing traffic across multiple Autonomous Systems (AS) further complicates adversarial efforts to track or analyze user behaviour, reinforcing the privacy of the communications.

Finally, multipath routing is closely related to developing mesh networks, which utilize a decentralized, non-hierarchical structure where nodes collaborate to relay traffic. While scalability concerns have historically limited the practicality of mesh networks for large user bases, recent innovations [26] offer potential solutions that may enable more widespread adoption of these resilient and flexible networks.

3

Related Work

This chapter provides a more detailed analysis of the concepts presented in Chapter 2, with appropriate references aligned with the dissertation's goals. Firstly, we will present the concept of Mixnets (primarily medium to high-latency anonymization networks) and how they enhance users' privacy in our solution. The following section will introduce the concept of K-Anonymization. Then, we will discuss how tunneling solutions work. Later on, we will revisit the subject of multipath strategies, presenting the relevant concepts and tools that can be applied to our solution.

3.1 Mixnets

Mixnets are a pivotal component in privacy-preserving technologies. They provide a robust defence against various forms of traffic analysis and enhance the confidentiality of online communications. These overlay networks, commonly known as Mix Networks, are purpose-built to enhance privacy by obscuring the origin of messages.

This concept was first introduced in 1981 [16] as part of an electronic mail system, using public key cryptography to hide whom a participant communicates and the communication content. Mixnets achieve this by routing data through a series of intermediary nodes, or "mixes", which introduce random elements like arbitrary jitter, shuffle, and reordering of messages, making it challenging for adversaries to correlate the incoming and outgoing traffic and thereby identify the source and destination of the data.

Despite mixing the messages on the network, Mixnets also employ encryption, enhancing security and ensuring that even if an adversary gains access to a mix node, the content of the messages remains secure and unreadable. This encryption often involves a cryptographic technique based on partial or random re-encryption, exemplified using the Sphinx packet format [23]. Furthermore, as messages pass through increasing nodes or hops within the data routing process, they undergo multiple layers of obfuscation. Thus, each node receives an encrypted message, then decrypts, batches, redirects, and sends it to the next hop [132]. Crucially, a proof of correctness is a component of this type of network, ensuring that the association between specific ciphertexts and plaintexts is kept

hidden [1].

Despite the concept of Mix Networks predating the development of Onion Routing by over a decade, their uptake has remained far behind for years. This historical limitation was primarily attributed to the relatively higher computational requirements, the introduction of additional latency in communication, and the lack of industrial-quality implementations [47]. However, the evolving landscape of internet censorship and the escalating concerns surrounding digital privacy have kindled a renewed interest in Mixnet-based anonymity networks. This resurgence in interest has culminated in the development of systems like Nym [30], Vuvuzela [113], PriFi [6], TARANET [18], Atom [65], XRD [64] and Loopix [93].

The recent revival of interest in Mixnets has also catalyzed several studies, including a novel simulation framework, a pioneering and pivotal tool, enabling researchers and engineers to evaluate different design options and trade-offs in Mix Networks comprehensively [13]. These recent advancements collectively hold the potential to usher in a new era of improved and enhanced implementations of Mix Networks and other anonymous systems, significantly contributing to the ongoing evolution of privacy-preserving technologies.

Among the latest strides in Mixnet evolution, Stadium [112], Karaoke [68], and Groove [7] leverage parallel Mixnets to prevent global adversaries from correlating users' communication activities and identifying their interactions. These architectures introduce pseudorandom connections between users and dead drops - ephemeral addresses where users exchange messages - while injecting noise traffic that confounds an adversary's observations [88]. While these systems showcase promising advancements in preventing correlation attacks, it is essential to recognize that they are each optimized for specific use cases like message broadcasting, private messaging, file-sharing, or future Internet architectures rather than for low-latency applications like web browsing or video streaming [88].

Understanding the design and operational principles of these parallel Mixnet systems is crucial for this thesis, enriching the conceptual foundation of the proposed solution. However, it is essential to note that these systems are not universally suitable for all use cases. For instance, most Mixnet architectures prioritize high-latency, high-privacy environments and may not meet the requirements of low-latency scenarios like video conferencing or interactive applications [88].

The following subsections present a detailed analysis of the Mixnet-based systems mentioned above, highlighting their unique features, strengths, and limitations.

3.1.1 Nym

Nym [30] is a privacy-focused infrastructure designed to enhance anonymous communication using a Mixnet-based architecture. It operates by routing messages independently through a network of mix nodes, which shuffle and re-encrypt the data at each hop, making it extremely difficult for even powerful adversaries to trace the origin or destination of a message. This approach prevents traffic analysis, where metadata such as the sender's

identity or message timing is typically exposed. Additionally, Nym obfuscates traffic patterns by introducing "dummy" messages - decoy packets that carry no payload and are discarded upon reaching their final destination.

The system is built for scalability and can be adapted for various applications, including secure messaging, anonymous file sharing, and cryptocurrency wallets. However, the network's growth might lead to competition among mix nodes for limited resources (representing node reputation). This competition could impose a practical limit on the number of new nodes an adversary could introduce as the network expands, as noted in recent research on Mixnets [47].

3.1.2 Vuvuzela

Vuvuzela [113] is a Mixnet-based messaging system that provides strong metadata privacy by using differential privacy techniques to inject noise into communications [91]. Messages are routed through multiple servers, where they are shuffled and encrypted before being forwarded. Each round of communication adds noise traffic, further obscuring the true source and destination of the messages. However, since Vuvuzela operates in rounds, users who are offline cannot receive messages, and all communications must pass through a single chain of relay servers [93].

Additionally, batch-based mixing systems like Vuvuzela, which rely on finite anonymity sets, tend to have smaller and theoretically unbounded anonymity sets compared to continuous-time mixes such as Nym, which offers more robust anonymity guarantees [30].

3.1.3 PriFi

PriFi (Privacy-Preserving Wi-Fi) [6] integrates Mixnets with Wi-Fi systems, enabling fast, anonymous communication over local networks. Unlike many traditional Mixnets, which suffer from high latency, PriFi is optimized for low-latency performance, making it suitable for interactive applications like real-time chat or video conferencing. While PriFi is resistant to traffic analysis and performs well for low-latency web browsing within LANs, it remains vulnerable to statistical disclosure attacks despite its enhanced privacy features [88].

3.1.4 TARANET

TARANET [18] is a low-latency Mixnet system. Traditional Mixnets often introduce significant delays, making them unsuitable for such applications; TARANET solves this by using traffic shaping techniques to ensure that data transmission is fast and secure. It maintains user anonymity while carefully controlling traffic flow and minimizing latency. However, its deployment poses challenges, as it requires changes at the ISP level. This could limit its initial impact until more widespread adoption is achieved.

3.1.5 Atom

Atom [65] combines the strengths of peer-to-peer networks and Mixnets to provide strong anonymity. By distributing communication across a decentralized peer-to-peer network and mixing messages through multiple nodes, Atom effectively thwarts traffic analysis; however, it relies on computationally expensive cryptography and requires messages to be routed through hundreds of servers in sequence, resulting in high latency [64]. It is designed explicitly for latency-tolerant, uni-directional anonymous communication, focusing on ensuring anonymity for the sender rather than both parties [93].

3.1.6 XRD

XRD [64] is a parallel Mixnet system that enhances anonymity by randomly rerouting messages through multiple nodes, adding a layer of security. This multi-path routing makes it much harder for adversaries to carry out correlation attacks, where they try to link a message's source to its destination by analyzing traffic patterns. Building on earlier Mixnet designs, XRD aims to reduce the latency typically associated with these systems, offering a more efficient and secure communication method for users who require strong privacy protections. However, like other batch-based Mixnets, XRD produces finite anonymity sets, which tend to be smaller - though theoretically unbounded - compared to continuous-time mixes like Nym [30].

3.1.7 Stadium

Stadium [112] is a parallel Mixnet system designed to withstand powerful adversaries, including those with global surveillance capabilities. It achieves this by splitting messages across multiple parallel paths, ensuring that no single route can expose the source or destination of the communication. Stadium also introduces "dead drops", temporary storage locations where users can anonymously send and retrieve messages. This additional layer of obfuscation makes tracing communications even more difficult. By combining parallel routing with dead drop techniques, Stadium offers strong anonymity and robust resistance to global traffic analysis.

However, like other batch-based Mixnets, Stadium produces finite anonymity sets, which are smaller - though theoretically unbounded - compared to continuous-time systems [30]. While Stadium is scalable, it does lack offline storage capabilities [93], meaning it does not provide mechanisms for message delivery when a user is offline for extended periods.

3.1.8 Karaoke

Karaoke [68] enhances anonymous communication by shuffling messages and introducing noise through ephemeral servers, which operate temporarily and constantly change, making it difficult for adversaries to track users over time. For communication

to occur, users must be online simultaneously, and their clients must send and receive messages during each round [7]. The system injects noise into communication patterns to prevent traffic analysis, ensuring that users' identities remain anonymous, even if some servers are compromised.

Karaoke offers strong anonymity guarantees while mitigating the risks associated with compromised individual nodes. Like other batch-based Mixnets, Karaoke produces finite anonymity sets, which are smaller - though theoretically unbounded - compared to continuous-time systems [30]. Additionally, similar to Stadium, Karaoke employs differential privacy in the process of adding noise to communication patterns, which introduces two key drawbacks: the probability gap between events can be sufficient for strong adversaries to exploit, and the system operates with a "privacy budget". This means users can deny a limited number of messages with strong guarantees, but as the privacy budget depletes, the level of privacy protection weakens. Once privacy levels drop, it remains unclear how to restore them effectively [64].

3.1.9 Groove

Groove [7] is a system designed to enhance anonymity in messaging and file-sharing applications by using pseudorandom connections and noise injection techniques to disrupt correlation attacks, which attempt to link senders and receivers based on traffic patterns. It also incorporates ephemeral addresses, or "dead drops", allowing users to exchange messages securely without revealing their identities.

While Groove supports messaging with low latency, high throughput, and horizontal scalability, it falls short in supporting complete asynchrony. The system offers a novel approach to temporary user disconnection (a form of partial asynchrony), allowing users to go offline for a limited number of rounds, after which they can no longer receive messages [98].

Moreover, during the setup phase, it uses an exhaustive pre-coordination method by generating circuits for all potential communication pairs (i.e., different nodes involved in communication). This setup enables message delivery without requiring additional coordination before each communication session. However, this approach significantly increases bandwidth consumption due to the need for pre-computed circuits for every potential pair of communicating nodes. Consequently, this method can limit the system's scalability by restricting the number of active participants (nodes) or necessitating powerful computational resources to maintain high throughput [58].

3.2 K-Anonymization

K-Anonymization and Mixnets, while distinct in their applications, both embody core principles of privacy preservation. Mixnets [16] strive to obfuscate the origin of messages by routing them through a series of intermediary nodes, manipulating the order

of messages, injecting delays, and using cryptographic techniques to ensure unlinkability between senders and receivers. In contrast, K-Anonymization [110] techniques protect individual identities by ensuring that each user or data record is indistinguishable from a group of K-1 others [125].

The concept of K-Anonymization is generally applied to data records where it is necessary to apply data transformations like generalization and suppression. These transformations hide sensitive attributes and make them difficult to tell apart [62]. The generalization phase replaces specific attributes with more generalized values, while suppression either removes or modifies data that could lead to the identification of individuals. These techniques are crucial for balancing data privacy and utility, as excessive anonymization may render the data less valuable for analysis.

While K-Anonymization is widely applied to various domains [118], such as Databases [61], Location Services [104], Social Networks [15] and in Big Data [4], it also finds a significant role in the context of network communication. This approach ensures that network participants remain camouflaged with a group of at least K-1 other participants. This makes it exceptionally challenging for adversaries to discern individual user activities or their origins. Notably, there are systems such as BriK [88], TIR [111], and TorKameleon [114] that leverage K-Anonymization to bolster network-level privacy for users and enhance the Tor Network.

3.2.1 BriK

BriK [88] was developed to enhance the privacy and anonymity guarantees of the Tor Network, particularly against adversaries with extensive monitoring capabilities, such as state-level or global ones. Its primary goal is to increase the number of possible active users using the same Tor bridge from one to a set of K-users, which strengthens the anonymity set associated with the source IP address of a Tor circuit from one to various users.

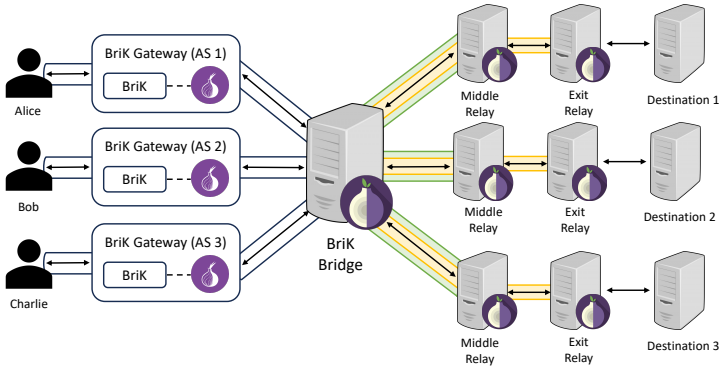


Figure 3.1: BriK diagram.

This Tor Pluggable Transport system consists of two main components: the BriK client and the BriK bridge. So, when a client wants to access a particular website over the Tor Network, it uses the BriK pluggable transport, where a BriK bridge acts as an ingress relay

for the user's Tor circuit. This unique bridge also intercepts the traffic from other K-1 users, which collaborate with the client by sending arbitrary data to the same bridge. This way, an attacker cannot determine which user sent what as the traffic undergoes shaping and coordination, making the data streams from all K users appear similar. Afterwards, the bridge discards the arbitrary data and only forwards the actual traffic to Tor. In order to ensure the original data is not distinguished, all traffic sent to the BriK bridge is encrypted and goes to the same modulation function.

By relying on this K-Anonymization technique to forward traffic to the Tor Network, an adversary that seeks to desanonymize the users will theoretically need to randomly guess the sender, resulting in a probability of 1/K of succeeding.

3.2.2 TIR

TIR [111], standing for Trusted Input Relay, is a solution that employs K-Anonymization by allowing multipath routing, where the traffic of one user can pass through the gateway (i.e. computer) of another user, all protected via TLS tunnels. This process was achieved by relying on various gateways, called TIR nodes, which connect to clients and then can send the traffic to entry nodes of the Tor Network. TIR relies on a network of ad-hoc community nodes (K-nodes) to route user traffic. Each user has its own local gateway, which can act as a proxy for other users, and users can also deploy additional proxies to enhance the network. TIR forwards requests, such as different HTTP requests or streams, through various paths formed by these different nodes, enabling multipath routing.

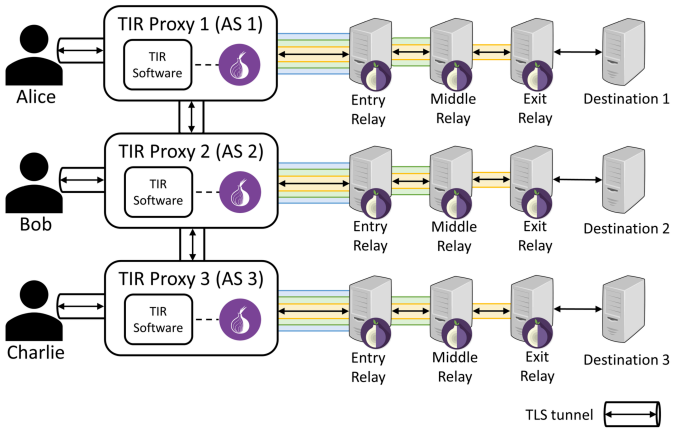


Figure 3.2: Tir diagram. Taken from [117].

TIR is not a fully integrated Tor Pluggable transport. As a result, it lacks the Tor-compatible protocols and specifications necessary to deploy TIR nodes as fully compatible Tor bridges. Moreover, contrary to BriK, it does not make all client traffic identical. Nevertheless, TIR can route traffic either through the Tor Network, via a pre-staged network, or as a standalone network separate from Tor.

This solution makes it more difficult for attackers to correlate traffic. If an adversary controls a Tor bridge, they will see the traffic coming from a TIR proxy, similar to how

BriK works, but this single proxy handles traffic from multiple users. Additionally, if an attacker is monitoring a user's network, that user may have an exposed TIR gateway that is also being used by other users, providing plausible deniability regarding the traffic exiting through that gateway. TIR has been tested against passive correlation attacks based on machine learning and deep learning models. Additionally, recent efforts extended the experimental assessment and analysis of the unobservability of TIR by demonstrating resistance against active watermarking traffic correlations or possible privacy breaks of the K-Anonymity sets of TIR users [90].

3.2.3 TorKameleon

TorKameleon [114] is a direct evolutionary step from the foundational principles laid out in TIR, where the focus was on advancing the concept of K-Anonymization through a multipath routing strategy. The core of this approach centres around establishing a network, often referred to as a random ad hoc flash mob group, consisting of K-proxies that engage in strategic communication. These K-proxies forward encapsulated user traffic, offering the flexibility of choosing between TLS tunnels or WebRTC video conferencing streams before they enter the Tor network.

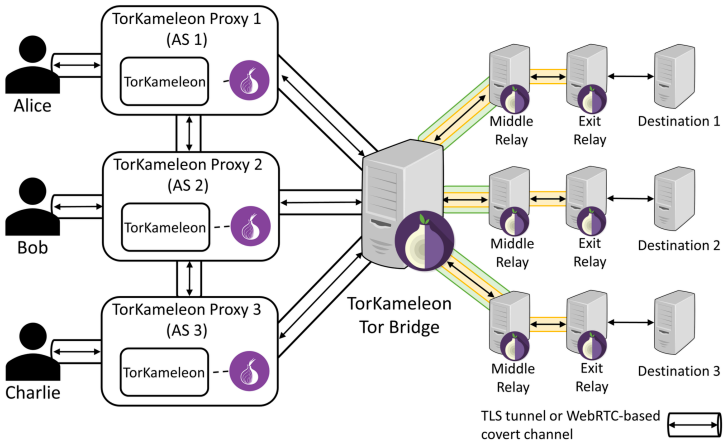


Figure 3.3: TorKameleon diagram. Taken from [117].

What sets TorKameleon apart from the other systems is the support for WebRTC-based covert channels (explained in Section 3.3.3.3) and its adaptability, offering users the freedom to choose how to deploy it according to their specific needs. It can operate independently as a standalone system, or alternatively, it can be seamlessly integrated as a Tor Pluggable Transport, aligning itself with the broader Tor Network while maintaining its core principles of multipath routing and K-Anonymization. However, like TIR, TorKameleon's client traffic does not pass a modulation function to make it all identical.

The effectiveness of the K-Anonymization mechanism in TorKameleon relies on the strategic placement of its proxies and bridges within different Autonomous System (AS) regions on the Internet, similar to TIR. Each proxy can be hosted on various platforms,

from local dedicated machines to cloud-based virtual private servers or Docker-based standalone containers.

3.2.4 Summary

The previous strategies are built upon the fundamental premise that there are K participants in the communication process. These participants collectively send traffic through the network, making it challenging for attackers to differentiate who sent the specific outgoing flow. However, for this technique to be effective, it is crucial to ensure that participants send traffic to the network. Failure to do so directly jeopardizes communications. Moreover, beyond ensuring the presence of K users, these solutions need to rely on the users to directly activate and synchronize themselves to ensure enough participants are active and sending traffic simultaneously, with the exception of BriK.

While TorKameleon offers a more robust solution than BriK and TIR by allowing traffic to be concealed within multimedia protocols such as WebRTC, making it difficult to detect hidden traffic, it becomes less resilient when there are few users in the network. So, if an adversary discovers that the WebRTC traffic is concealing some message, it is quickly possible to find out where it comes from. On the other hand, BriK offers unique features that are not found in either TIR or TorKameleon. Notably, BriK is the only solution that provides automatic synchronization between clients, ensuring a stronger level of privacy.

These challenges highlight the delicate balance that K-Anonymization systems must strike between ensuring robust privacy and security while maintaining network participation, synchronization, and resistance to advanced correlation techniques.

3.3 Tunneling Solutions

Secure and efficient data transmission across networks is essential on the Internet, where privacy is becoming increasingly vulnerable. This need is evident in various contexts, whether for private communications, enabling remote access to resources, or even ensuring the confidentiality of sensitive information.

One key element of network security is tunneling solutions, which provide a secure and reliable means of encapsulating and transmitting data across potentially untrusted networks. While the concept of tunneling is not new, it has become more critical due to evolving challenges. Notably, censorship regimes continuously adopt and deploy state-of-the-art techniques to detect and analyze Internet communications. As a response to these pressing challenges, tunneling solutions have witnessed continuous advancements and innovations.

3.3.1 Standard Protocol-based Tunneling

Standard Protocol-based tunneling solutions offer a versatile approach to secure data transmission by creating a virtual, isolated path or "tunnel" within a larger network

infrastructure, where data can traverse while being concealed within the tunnel. In addition to ensuring compatibility with a wide range of network devices and systems, these solutions are harder to detect and block by censors, as they blend in with regular traffic, making it difficult to impose restrictions without causing collateral damage to legitimate network services.

Among the key standard protocols used for tunneling, Secure Sockets Layers (SSL) and Transport Layer Security (TLS) are critical for securing web communications. They establish encrypted connections between the web servers and the clients, safeguarding data privacy and integrity. Moreover, Secure Shell (SSH) is a robust solution for secure remote access and file transfers. It establishes encrypted tunnels for data communication, ensuring that sensitive information remains protected during the session. Virtual Private Network (VPN) protocols, such as OpenVPN, IPsec, and L2TP, create encrypted connections between devices and networks. These protocols are highly versatile and allow secure data transmission over public networks like the Internet. In addition to these standard protocols, recent studies in the literature have begun to explore the use of QUIC [67] as a protocol for encapsulation, demonstrating its potential for tunneling by encapsulating and securing network traffic within a QUIC connection. Moreover, recent research showcases that HTTP/3 requests, which use QUIC as the underlying encrypted transport, are less frequently blocked than traditional HTTPS requests, or sometimes even not blocked at all [35].

However, in regions where censorship and surveillance are prevalent, adversaries actively control or block/restrict standard protocols like VPNs, SSH, and SSL/TLS [53, 128], so users may need to use different tools capable of remaining unblocked, as blocking the carrier protocol might lead to potential collateral damage. To address these challenges, users may turn to alternative techniques, such as Media Protocol-based Tunneling solutions, which focus on obfuscating network traffic by embedding it within media protocols, making it difficult for censors to detect and block.

Censors recognize that attempting to block or restrict media protocols would essentially mean blocking a vast majority of the internet content. So, given this scenario, media protocols remain largely untouched by censors as they understand that impeding or eliminating these protocols would significantly disrupt the flow of online information and services but also ignite discontent among Internet users. The potential backlash from such actions might fuel an unprecedented wave of resistance, triggering a massive protest against censorship and surveillance practices. Furthermore, from an economic standpoint, certain companies within the affected countries rely on these tools to communicate with other businesses and facilitate their operations. Thus, maintaining unrestricted access to information extends beyond the population's well-being to encompass the country's economic interests. As such, it is more feasible for censors to target specific applications like YouTube or Zoom than to block an entire protocol.

3.3.2 Media Protocol-based Tunneling

Media Protocol-based Tunneling solutions emerge as innovative strategies for covertly conveying information in secure data transmission. These solutions involve encapsulating network traffic within media protocols, using audio and video streams from popular media platforms like Skype or YouTube or even entire protocols such as WebRTC, and providing a discreet channel for communication.

Behind this approach is the insight that a big percentage of the traffic online comprises video streaming [42]; as such, this prominence makes media streaming an ideal cover for covert data transmission. These solutions strive to blend data seamlessly with regular traffic, harmoniously integrating the data into the regular traffic, attempting to make censors incapable of distinguishing between conventional data streams and those carrying covert channels [9, 44]. Therefore, these systems aim to pose a substantial challenge for censors to block without causing significant collateral damage [38].

Media tunneling solutions aim to provide unobservability and unblockability and ensure versatility, resilience, flexibility, and adaptability. This reflects a focus on seamless integration, robust operation, bidirectional communication, and the secure transmission of various data types [12].

The main objective of these solutions is to evade detection and censorship by disguising traffic patterns within legitimate network flows. However, discrepancies may arise if they fail to closely resemble regular traffic patterns, allowing censors to detect irregularities. This detection can occur through statistical analysis or machine and deep learning models that compare outgoing and incoming traffic. These methods often analyze traffic features such as metadata, packet length, and arrival times to identify deviations from normal behaviour [12].

Some examples of Media Protocol-based Tunneling solutions include FreeWave [52], a tool that allows clients to browse the uncensored web, leveraging the cover of VoIP communication for covert data transmissions. Another solution is Facet [70], a tool that allows users to watch videos from websites by tunneling these videos into seemingly innocuous Skype calls. CovertCast [79] presents a unique approach by transforming website content into videos, which are then broadcasted through a popular video streaming service. This method not only disguises the nature of the transmitted data but also leverages the ubiquity of video streaming, contributing to covert communication seamlessly integrated into regular online activities. In addition, DeltaShaper [10] stands out for enabling the covert transmission of arbitrary TCP/IP traffic by using video conferencing services as carriers (e.g. Skype). Lastly, VoiceOver [57] introduces an audio-based multimedia protocol tunnel over Skype calls, employing a sophisticated design based on a generative machine learning model. This design restricts covert data transmission to align with the timing properties learned from real two-person conversations, ensuring the covert channel mimics the natural flow of genuine interactions.

Despite these advanced capabilities, most of the solutions mentioned above fall short

of exhibiting the necessary characteristics to effectively bypass Internet censorship and provide robust unobservability, with research indicating that these channels are vulnerable, as advanced methods can uncover the majority of them while maintaining comparatively low false-positive rates [9, 44].

Despite Media Protocol Tunneling solutions, other techniques take a distinctive approach, and instead of tunneling, they rely, for example, on Media Protocol Mimicking techniques. They aim to replicate standard media protocols' behaviours and statistical properties, enabling covert data transmission. SkypeMorph [82], for instance, specializes in mimicking the behaviours of Skype traffic. By precisely emulating the patterns of regular Skype communication, SkypeMorph endeavours to provide covert data transmission that integrates with genuine Skype traffic, making it arduous for censors to distinguish between the two. CensorSpoofer [121], on the other hand, focuses on VoIP traffic; employing advanced techniques, CensorSpoofer endeavours to mirror the statistical properties and characteristics of genuine VoIP transmissions. However, these types of mimicking solutions are inherently complex and frequently struggle to accurately mimic the behaviour of media traffic [51].

3.3.3 WebRTC-based Tunneling

WebRTC-based tunneling is a specific implementation of media protocol tunneling solutions. The media protocol used to tunnel the hidden data is WebRTC, which stands for Web Real-Time Communication. This type of solution harnesses the inherent features of the WebRTC protocol to facilitate covert data transmission within web-based communication channels. WebRTC itself is a collection of APIs (Application Programming Interfaces) and protocols that facilitate real-time communication over the web. It enables direct peer-to-peer communication between (most of the) browsers or applications, incorporating functionalities like voice calling, video conferencing, and file sharing without the necessity of plugins or external software [107].

In order to ensure the confidentiality, integrity, and authenticity of communication sessions, WebRTC incorporates robust security measures. Its security features are multi-layered, addressing various potential vulnerabilities. Encryption lies at the core of WebRTC's security architecture, employing the Secure Real-Time Protocol (SRTP) for encrypting audio and video streams, ensuring that transmitted data remains confidential and protected from unauthorized access or eavesdropping attempts. Moreover, WebRTC leverages authentication protocols like DTLS-SRTP to authorize legitimate participants, ensuring that only authorized entities engage in communication. It uses STUN and TURN servers for network traversal, facilitating communication across devices behind restrictive configurations of firewalls or NATs. Lastly, secure signalling mechanisms are integral, enabling the establishment of connections while exchanging session details and encryption keys [29]. WebRTC does not manage the signalling mechanisms itself; instead, the responsibility for handling the signalling protocol is delegated to the application.

This section delves into the workings of various examples of such solutions, exploring their unique features and their valuable contributions to online privacy.

3.3.3.1 Protozoa

Protozoa presents a novel approach that capitalizes on WebRTC's functionalities to enable robust and censorship-resistant internet communications, allowing users to exchange information discreetly and securely while offering reasonable performance for typical internet communications and resilience against traffic analysis [11].

At its core, Protozoa exploits the WebRTC framework's fundamental design, particularly its ability to establish direct peer-to-peer connections between web browsers without intermediary servers. By embedding IP packets within WebRTC multimedia streams, making it challenging for censors to detect or block such covert transmissions.

In order to establish a covert channel, a user initiates a video call via a widely used WebRTC streaming service, such as Whereby, connecting with a trusted party outside the censored region. On the free side, this trusted party operates as a proxy for the Protozoa system, routing the user's traffic through the proxy to its final destination. Once connected, the user can browse the Internet freely, with the WebRTC connection serving as a covert channel to bypass censorship. The efficiency of this solution lies in making it difficult to distinguish between Protozoa communications and normal WebRTC traffic, resembling typical audio-video calls.

While proven effective against state-of-the-art traffic analysis techniques [9], Protozoa exhibits certain limitations [8]. Specifically, it is vulnerable to potential man-in-the-middle attacks when WebRTC applications rely on WebRTC gateways to mediate user connections. In such cases, an adversary controlling the WebRTC gateway can inspect the media streams and easily detect covert payloads within the communication, because they carry arbitrary covert data that does not conform to media codec specifications [40].

3.3.3.2 Stegozoa

Stegozoa, an evolution from the foundations laid by Protozoa, introduces an advanced WebRTC-based tunneling solution to achieve better unobservability, especially in scenarios where malicious gateways are present within the WebRTC application. This system integrates steganography techniques to conceal information within the video stream content, effectively preventing detection while allowing censors to inspect the video payload without uncovering the hidden data [40].

This solution enhances the security measures employed by Protozoa because steganography techniques are used to ensure that the video remains indistinguishable from a regular video call, even if the censor is observing the unencrypted video. However, this emphasis on security comes at the cost of reduced throughput, dropping from 1.4Mb/s (Protozoa) to 8.2Kb/s, which restricts its practical usability. Like Protozoa, Stegozoa is not

integrated into widely known anonymity networks like Tor, which also limits its broader application.

Protozoa and Stegozoa find their strength within controlled experimental setups designed to operate effectively within the user's network environment. These solutions excel in scenarios where direct control and configuration are feasible, allowing for efficient and secure communication under specific conditions. However, their potential integration within anonymity networks is limited, implying a need for further adaptations before incorporating them. Additionally, these solutions pose significant challenges in deployment and local testing, indicating that practical application might require overcoming substantial obstacles.

3.3.3.3 TorKameleon

As mentioned in Section 3.2.3, TorKameleon [114] stands out as a novel solution based on K-Anonymization with flexible encapsulation options, offering the choice between TLS or covert WebRTC-based channels, which aligns with the scope of this section. Unlike Protozoa and Stegozoa, which require changes to browser source code or the injection of additional code, TorKameleon relies solely on the APIs provided by WebRTC. This makes it a much lighter and more practical solution for deployment.

The encapsulation process in TorKameleon involving WebRTC begins by splitting the video stream into discrete audio and video tracks. Covert data is then embedded within specific video frames, camouflaging the data within the media stream to make it indistinguishable from regular WebRTC traffic, effectively evading detection by censors. These covert-data-embedded frames are transmitted to the bridge, decapsulated, and the covert data is extracted.

TorKameleon's WebRTC encapsulation has undergone meticulous testing against active correlation attacks. Notably, while it provides performance comparable to the previously mentioned systems [11, 40], it offers a highly configurable solution that is fully compatible with Tor and resilient against both active and passive correlation attacks, making it easily deployable in real-world scenarios.

3.3.3.4 Snowflake

Snowflake [14] presents an innovative approach to circumventing Internet censorship by leveraging WebRTC for peer-to-peer proxy communications. Unlike TorKameleon [114], which embeds covert data within media streams, Snowflake uses data channels. Embedding data in media streams, as done in TorKameleon, may offer stronger obfuscation due to the possible lack of applications that use data channels. This system utilizes a dynamic collection of volunteer proxies, termed "snowflakes", which are ephemeral and challenging for censors to track or block due to their transient nature and widespread geographic distribution.

The core mechanism of Snowflake involves the establishment of temporary, browser-based proxies that connect to the client via WebRTC data channels, facilitating the transfer of censored content to a bridge and bypassing censorship mechanisms. The proxies operate on an ad-hoc basis, launched when users open their browsers and disappear when they close them. This dynamic behaviour creates an ecosystem difficult for censors to block, as the proxies constantly appear and disappear, making it challenging to track or shut down the system effectively. This model capitalizes on the ubiquity of WebRTC-enabled browsers to create a robust pool of proxies without requiring participants to undertake complex configurations or commit long-term resources. Snowflake's architecture enables a high degree of flexibility and scalability. The system's resilience against active and passive censorship attempts is enhanced by its reliance on WebRTC data channels, which blend seamlessly with everyday internet traffic. This system has been rigorously tested in various censorship-heavy regions, showing remarkable efficacy in enabling uncensored access to the internet.

3.4 Traffic Splitting Solutions

There has been growing interest in multipath strategies to enhance user privacy by distributing traffic across multiple network paths in recent years. These techniques are increasingly important for mitigating attacks such as website fingerprinting, where adversaries attempt to infer user activity by observing traffic patterns, even when the traffic is encrypted.

A notable system employing this strategy is CoMPS (Connection Migration Powered Splitting) [120], which utilizes the connection migration features of protocols like QUIC and WireGuard to route traffic across multiple paths mid-session dynamically. This system is particularly relevant to this thesis because it comprehensively evaluates traffic splitting in terms of throughput and resistance against website fingerprinting attacks. Moreover, the insights drawn from CoMPS provide valuable context for understanding different configurations employed in traffic-splitting. This topic will be further discussed in Section 6.4.

Other systems focus on mitigating website fingerprinting attacks but rely on different mechanisms rather than connection migration, such as proxy setups and Multipath TCP (MPTCP). For instance, TrafficSliver [28] is a system that leverages a proxy setup to split traffic across multiple paths, while HyWF [50] uses MPTCP to distribute traffic across different paths. However, using connection migration, CoMPS provides a more flexible and scalable approach by being deployed with any server supporting these features.

It is important to note, however, that CoMPS does not encapsulate or disguise traffic like some other systems do. Systems like TorKameleon and Stegozoa rely on encapsulation techniques to obfuscate traffic by embedding it within media streams, making it harder for adversaries to detect the nature of the traffic. In contrast, CoMPS enhances privacy through multipath splitting, relying on the underlying protocol to fragment traffic across

different routes. While this approach excels in distributing traffic and decreasing the amount of metadata an adversary can obtain, it does not provide the same degree of traffic obfuscation as systems that actively disguise the traffic's nature.

3.5 Summary

This chapter encompasses diverse concepts and solutions found in the related work and literature, offering an extensive analysis of the various approaches to circumventing Internet censorship. Central to this discussion is the recognition that adversaries actively employ traffic analysis attacks [37, 41, 77, 86, 87, 95, 127], leveraging machine learning [5, 9] and deep learning techniques [45, 46, 73, 85, 89] to compromise anonymization networks like Tor [32]. In response, this chapter highlights the significance of K-Anonymization techniques [88, 110, 111, 114, 125], tunneling solutions [9–11, 40, 44, 52, 57, 70, 79, 114] and dynamic multipath routing strategies in bolstering user anonymity.

K-Anonymization: Although K-Anonymization solutions like BriK [88] and TIR [111] are significant, they typically lack encapsulation methods and require complex user coordination and synchronization, posing potential operational complexities.

Tunneling & Tradeoff of Security and Latency: Standard [31, 43, 100, 130] and non-WebRTC media-based [10, 52, 57, 70, 79] tunneling solutions, although promising, often falter against traffic analysis attacks [9, 44]. On the other hand, WebRTC-based tunneling solutions, exemplified by Protozoa [11], Stegozoa [40], TorKameleon [114] and Snowflake [14], demonstrate the ability to uphold desired security properties with reasonable latency. Crucially, they enable bidirectional communication and end-to-end tunneling of arbitrary IP traffic.

Integration with Anonymity Networks: Protozoa [11] and Stegozoa [40] were developed in controlled laboratory environments and as standalone tools; as such, they lack integration with existing networks and are challenging to deploy. In contrast, TorKameleon [114] is integrated with the Tor Network [32] via a pluggable transport module and newly designed bridges. Similarly, Snowflake [14] is also deployable through a bridge in Tor.

Multipath Strategies: TorKameleon [114] is a good inspiration base, and due to the embedment of data within media streams instead of data streams, it can potentially provide better resistance against censorship than Snowflake [14]; nonetheless, some limitations arise when interacting with Tor [32]. In particular, the fixed multipath approach for each connection or data stream contrasts with a dynamic multipath approach, as seen in systems like Loopix [93], which enables per-message routing and potential rerouting along diverse paths. The Tor's reliance on TCP and the absence of advanced multipath features

lower the potential performance of the network and significantly impact latency-sensitive activities such as video streaming.

Traffic Splitting: Traffic splitting is an approach to enhance privacy by distributing data across multiple paths, but most implementations rely on specific configurations or protocol features, which can limit adaptability. For instance, systems like CoMPS use connection migration within protocols like QUIC to enable path distribution [120], while others employ static routes, reducing flexibility. Moreover, many of these solutions, such as Conflux [3] and TrafficSilver [28], focus solely on traffic splitting without adding encapsulation, which can leave relevant metadata exposed to sophisticated traffic analysis.

Opportunity: Considering the points mentioned, there is an opportunity to create a highly configurable solution that combines the strengths of various existing systems. By leveraging TorKameleon's proven resilience against state-of-the-art correlation attacks [114], such a system could integrate dynamic multipath routing of arbitrary IP traffic, enabling packets to be rerouted through distinct paths. Moreover, we could further enhance the privacy of the system by creating a novel approach: Multipath Multi-encapsulation, where the paths could be established using covert channels embedded within TLS, QUIC, or WebRTC-based encapsulation, with the possibility of each segment of the circuit being encapsulated in a different type of protocol, further obscuring traffic characteristics aiming at making the detection and correlation by adversaries significantly more challenging.

Proposed Solution: The envisioned network based on dynamic multipath and multi-encapsulation routing would serve as a novel, purpose-built solution, building on the strengths of TorKameleon while addressing key limitations of the Tor Network, particularly its lack of traffic splitting. By enabling dynamic and multi-encapsulation traffic splitting, this solution aims to provide greater resilience against traffic analysis techniques like website fingerprinting, which often exploit single-path systems like Tor. At the same time, encompassing equally important performance metrics such as reasonable latency and throughput, for everyday internet activities.

	BriK	TIR	Protozoa	Stegozoa	TorKameleon	Snowflake	COMPS	**Proposed Solution**
K-Anonymization	✓	✓	x	x	✓	x	x	**x**
Good Throughput and Latency	✓	x	✓	x	✓	✓	✓	**✓**
Video Streaming Tunneling *(WebRTC)*	x	x	✓	✓	✓	✓	x	**✓**
Multiple Encapsulation Options	x	x	x	x	✓	x	x	**✓**
Dynamic Multipath w/ Traffic Splitting	x	x	x	x	x	x	✓	**✓**
Multipath Multi-encapsulation	x	x	x	x	x	x	x	**✓**
Standalone System	x	✓	✓	✓	✓	✓	✓	**✓**

Table 3.1: Comparing techniques across Internet Circumvention Systems.

According to the above discussion, in Table 3.1, we briefly compare the Internet Circumvention Systems mentioned in this chapter according to the techniques employed. The

✓symbol indicates that the system performs well in the corresponding aspect, while the X symbol signifies that the system lacks or performs poorly in that area.

4

System Model and Architecture

In this chapter, we detail the system model and architectural framework of MIRACE, beginning with an introduction to the system's proposal. We then articulate the design goals and delineate the proposed solution, setting the stage for an in-depth exploration of the system's architecture and components. Following this, we present the adversary model, discussing in detail the specific threats it poses and the countermeasures designed to mitigate these risks. The chapter concludes with a comprehensive summary that synthesizes the key points discussed.

4.1 MIRACE Proposal

As shown in Section 1.2, the current landscape of online surveillance and censorship has highlighted the necessity of preserving user privacy and anonymity. Existing solutions, exemplified by the Tor Network, face increasing scrutiny and challenges in maintaining adequate levels of anonymity without compromising performance [25]. With adversaries intensifying efforts to compromise user privacy, there is a pressing need for a novel solution that can effectively stand state-of-the-art censorship techniques, ranging from passive (Section 2.2.1) to active attacks (Section 2.2.2), while maintaining reasonable latency. Therefore, we have developed MIRACE, a censorship-resistant tunnelling tool that provides online anonymity and privacy through dynamic multipath routing at the packet level and multi-encapsulation within the same circuit. This means traffic is split and sent across different mixed paths using advanced and flexible encapsulation options.

The main goal is to achieve **unobservability**, **unblockability** and **reasonable performance**, which involves preventing the inference of covertly transmitted data content and drastically decreasing the likelihood that adversaries will not control every pathway simultaneously. Moreover, our solution intends to prevent a censor from blocking it without causing adverse collateral effects, such as blocking TLS, QUIC, WebRTC communication, or access to public cloud providers. Lastly, it also aims to optimize resource utilization and allow overall reasonable performance, allowing its use for low-bandwidth Internet tasks and, ideally, for accommodating streaming services.

4.2 System Design Model

This section provides a comprehensive overview of the system model and design considerations for MIRACE. We begin by outlining the key design goals that have guided the system's development, followed by the detailed architecture of MIRACE, including a description of the overall system architecture. Then, we will delve into each component in detail, including how they contribute to the system.

4.2.1 Design Goals

The key idea of MIRACE is to provide stronger anonymity guarantees to Internet users who face censorship or want to protect their privacy online. To achieve that, we combine Onion Routing, which adds multiple layers of encryption to packets and forwards them through multiple nodes while keeping the sender anonymous, with a dynamic multipath routing strategy at the packet level rather than the traditional stream or request level. Those paths can be selected based on security and performance metrics, prioritizing better latency or security depending on the user's needs. Moreover, we allow diversity in the choice of encapsulation mechanism by giving the user a choice between a secure TLS or QUIC channel and a WebRTC-based covert media streaming channel. This makes our system traffic resemble regular internet traffic, complicating detection and filtering by adversaries. Furthermore, using different encapsulation types across distinct paths hinders correlation between paths, enhancing resistance to adversarial tracking and analysis.

The design of the solution is driven by the following objectives:

- **Generic Functional Solution:** The solution must be a functional, generic system that any user can use to access the internet securely and privately. It must be easy to use and require minimal configuration to work effectively through a simple script.

- **Unobservability:** The solution is designed to be unobservable, making it difficult for adversaries to detect or block its use. It encapsulates traffic within conventional internet protocols such as TLS, QUIC, and WebRTC to achieve this, making it challenging for adversaries to identify. In addition, censors should not be able to identify the sender or link a specific output stream to a particular user. For this purpose, MIRACE employs dynamic multipath routing (by frequently changing paths, it becomes exceedingly difficult for adversaries to track or intercept data) and even traffic padding. These methods have demonstrated the most effective outcomes in countering machine learning models for traffic correlation, which includes advanced modern deep learning tools described in Chapter 2.2.1.

- **Unblockability:** The solution is designed to bypass censorship and filtering mechanisms without causing significant disruptions to the global internet infrastructure, such as blocking TLS, WebRTC communication, or access to public cloud providers.

- **Performance:** The solution must be performant, ensuring users experience the minimal latency possible for such security considerations and reasonable throughput. Moreover, it should be parameterizable based on security and performance metrics. The user can define whether he wants to change the paths more frequently or not, based on the time that has already passed or on the number of packets already sent. Lastly, it should be able to handle a large number of users simultaneously without compromising performance.

4.2.2 System Model Overview

The solution is designed around a distributed architecture comprising several key components that ensure user data's integrity. The architecture operates on a collaborative network model, facilitating secure communication between participating nodes with TLS, QUIC tunnels or WebRTC-based covert media streaming.

At its core, the architecture revolves around two types of voluntary nodes: MIRACE's relays and local gateway nodes, as shown in Figure 4.1. The MIRACE's relay nodes serve as the network's backbone, functioning as simple relays that forward packets to subsequent network nodes or as aggregators (a subtype of relay nodes, i.e., exit nodes) that reassemble split flows into their original form and direct them to their destination. Meanwhile, the local gateway nodes, run locally on each user's device, transmit user traffic and receive the corresponding responses via the relays.

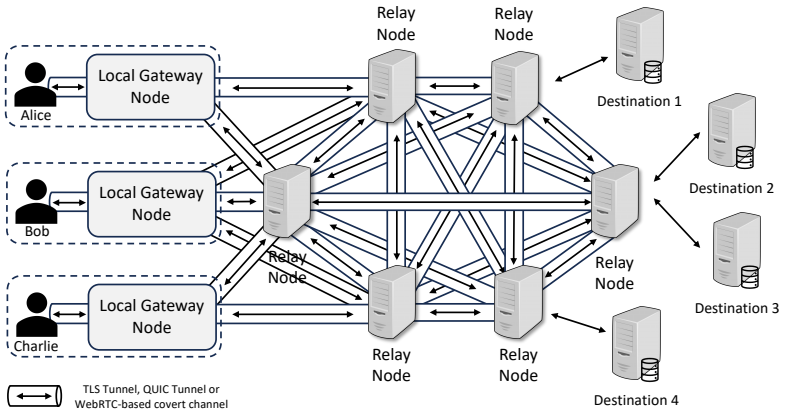


Figure 4.1: Proposed Solution diagram.

In Figure 4.2, we illustrate a scenario where a client seeks to access a web server designated as "Destination 1". The diagram depicts multiple pathways along which the client's traffic is distributed. This diagram is key to understanding how traffic is split and managed within our proposed system to enhance security.

To ensure the integrity and sequential order of the transmitted data, a critical architectural requirement is that the exit node for each of these divergent paths remains identical. This stipulation is crucial because it allows for proper reassembly and synchronization of the data packets at the destination despite their disparate routes through the network. By

maintaining a consistent exit node for all traffic paths, the system ensures that all segments of the data arrive at "Destination 1" in their original order, thereby preserving the flow and coherence of the information.

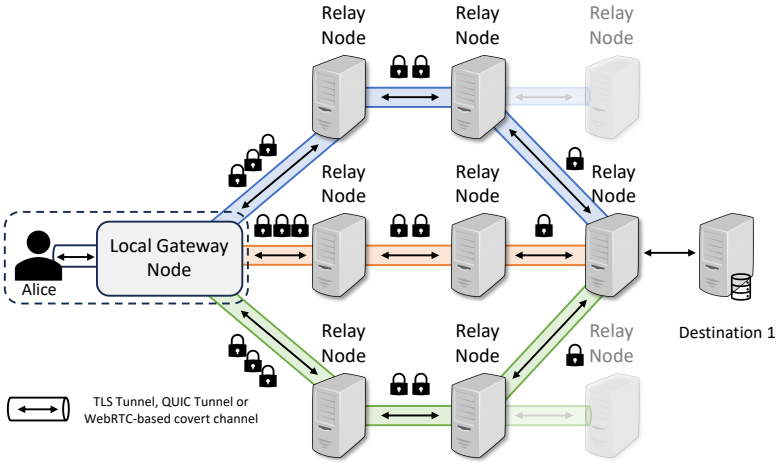


Figure 4.2: Overview of traffic splitting on one long-lived connection to a web server.

As aforementioned, the traffic can be split based on predefined heuristics that dictate the frequency of changing the paths. In specific scenarios, users may opt for a strategy of changing paths less frequently, which results in traffic not being split across multiple routes, as illustrated in Figure 4.3.

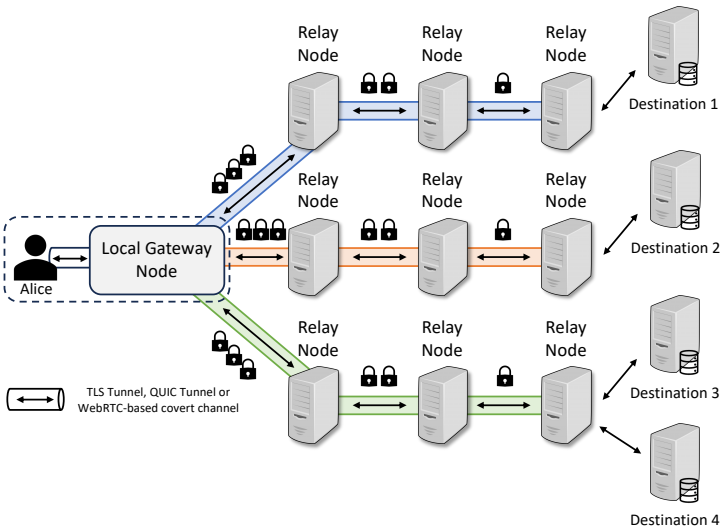


Figure 4.3: Overview of traffic splitting on various short-lived connections to web servers.

This strategy of changing the paths less frequently can lead to significant security trade-offs. With fewer paths in use, the system's ability to withstand traffic analysis attacks, including correlation attacks and website fingerprinting, is weakened. This makes it easier for a potential censor or malicious actor to carry out traffic correlation attacks, as controlling fewer nodes becomes more feasible and effective in compromising user anonymity.

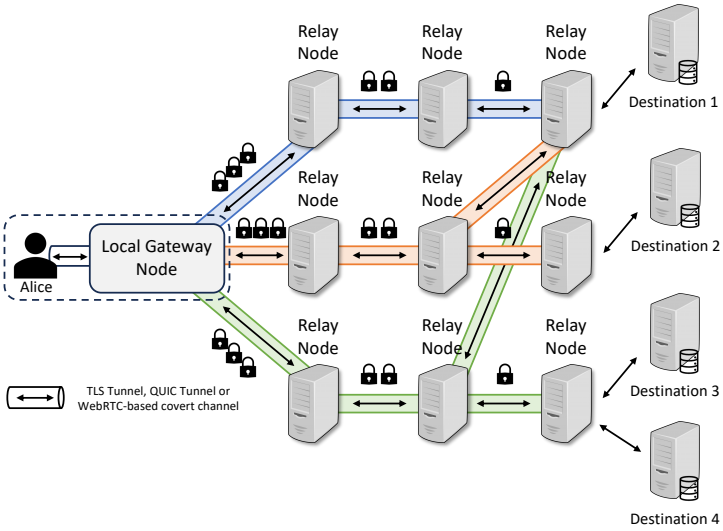


Figure 4.4: Overview of traffic splitting on various short-lived connections with one long-lived to web servers.

Conversely, a more comprehensive depiction of the system's operation is provided in Figure 4.4. This figure illustrates that long-lived incoming connections need to utilize the same aggregator across various paths while others do not; if the paths are changed infrequently and the connection duration is short, the packets of that particular session end up being routed exclusively through one circuit. This operational dynamic is similar to that observed in systems like the Tor Network, where a single path is used for the duration of a connection. However, in our system, this can be fixed by simply reducing the threshold for path switching, ensuring that even these short-lived connections are forwarded through multiple paths, enhancing security.

These scenarios highlight the crucial balance between performance and security in network architectures. While opting for fewer path changes (when prioritizing routes with lower latencies) may enhance performance, as it avoids paths that might act as bottlenecks due to higher latency, it also significantly increases the system's exposure to privacy risks. Therefore, the decision to limit path changes must be carefully managed, with a clear strategy for mitigating potential vulnerabilities, especially in environments where maintaining privacy is of utmost importance.

4.2.3 Covert Channels and Traffic Diversity

Ensuring secure and anonymous communication is a critical priority in the proposed distributed network architecture. This subsection delves into integrating advanced communication protocols - specifically TLS tunnels, QUIC tunnels, and WebRTC-based channels - as potential means for facilitating covert communications. Each of these protocols plays a vital role in diversifying traffic, a key strategy for enhancing security and obfuscating user activities from potential surveillance and censorship.

TLS (Transport Layer Security)

TLS, a cornerstone for secure communication over a computer network, is not just a security measure in our architecture. It is also used strategically to disguise through covert channels. By encapsulating traffic within TLS, data packets blend in with other encrypted traffic, making them harder to distinguish from regular secure communications, thereby camouflaging the encrypted messages from our system within the broader flow of encrypted data. This method makes it extremely difficult for network monitors and censoring agents to distinguish these packets from regular internet traffic.

QUIC (Quick UDP Internet Connections)

Developed by Google, QUIC is a modern transport layer protocol that operates over UDP. Its adoption in our system brings several advantages crucial for maintaining the confidentiality and fluidity of communications. QUIC's ability to multiplex multiple data streams over a single connection enhances the speed and reliability of data transfers and introduces a layer of traffic obfuscation that complicates potential interceptive efforts. The protocol's inherent encryption standards bolster this by providing robust security across all transmitted data. Additionally, QUIC reduces connection establishment times and improves performance in conditions of packet loss, which are vital for maintaining a seamless and secure user experience in environments where network conditions are highly variable.

WebRTC (Web Real-Time Communication)

WebRTC enables direct real-time communication between browsers and devices without additional plugins or configurations. Within our architecture, WebRTC is pivotal for setting up covert channels through media streaming. By embedding data into media streams, WebRTC channels can transmit information in ways that are exceedingly difficult to detect or intercept. This feature is especially crucial in regions or scenarios where standard communication methods, such as conventional network traffic, might be more susceptible to censorship or intensive surveillance.

Summary: The strategic integration of TLS, QUIC, and WebRTC into our network's architecture is designed to address the dual challenges of performance and privacy. By leveraging these protocols, we not only enhance the robustness of data transmission against external threats but also significantly increase the complexity of traffic patterns, making it much harder for adversaries to analyze and trace communications. This comprehensive approach ensures that our system remains resilient against various forms of network surveillance and censorship, thus maintaining the integrity and confidentiality of user communications.

4.3 Adversary Model

This section provides a detailed model of the adversary, exploring their capabilities and motivations. By examining their potential tactics and strategies, we aim to understand the threats they could pose to our solution. This analysis forms the foundation for creating effective defence mechanisms and counterstrategies to prevent their advances and ensure the resilience of our system.

We categorize adversaries into three main groups based on their capabilities and the extent of their influence. **State-level adversaries** are those capable of controlling one or very few Autonomous System (AS) regions, typically regimes in smaller countries or local government agencies. Next, we present two subcategories of Global level with different capabilities, **Large-Scope Adversaries** and **Continental Adversaries**. Broadly speaking, these adversaries can control multiple AS regions, which might stem from collaborations with other entities such as Internet Service Providers (ISPs) or governmental organizations or from the sheer extent of their geographic dominance. Also, we broaden this category to large corporations that control significant parts of the internet infrastructure, particularly those that manage substantial portions of data flows and have extensive capabilities in hosting services, which may be problematic if they control and manage a large portion of the nodes of an Anonymization Network, as they have the resources available to deanonymize the users if wanted or needed. Finally, the **Omnipresent Adversary** represents the most challenging level, with the capability to control every internet connection, every device, and every AS region across the globe.

For **State-Level Adversaries**, the potential threats of deploying compromised nodes are usually confined to a specific geographic or network area, which may limit the overall impact but still pose significant localized risks. When these adversaries engage in both deploying nodes and monitoring them, their capability to disrupt or spy is considerably enhanced, presenting a complex challenge for network defence. These adversaries can observe, capture, analyze, and perform correlation attacks, including circuit fingerprinting and timing attacks, but cannot crack the cryptography that protects secure communications. They can also inject watermarks to conduct active correlation attacks and compromise user anonymity through intersecting K sender-receiver sets.

In the case of **Large-Scope and Continental Adversaries**, while they can perform the same actions as state-level adversaries, their ability to launch more powerful attacks is heightened due to their control over larger areas and potentially having more points of presence. Large-Scope Adversaries typically influence multiple regions within a country or across several countries, often through collaborations with ISPs and other infrastructure providers. Continental Adversaries, however, wield influence across a broader, continent-wide scale, giving them access to significant portions of critical infrastructure, which could increase their capacity for powerful attacks. This influence is not limited to the malicious deployment of Sybil nodes; it extends to access across multiple regions, such as

collaborations with ISPs in multiple regions, significantly increasing their threat.

Lastly, it is important to clarify that our threat model does not base its assumptions on the presence of an **Omnipresent Adversary**. Such an adversary, who engages in both deploying and monitoring nodes, represents the ultimate threat model. While providing a comprehensive understanding of potential risks, this hypothetical scenario is considered impractical for realistic defence strategies. Instead, our focus is primarily on understanding and mitigating the risks of State-Level and Large-Scope/Continental Adversaries.

We also leverage the idea that historically, censors have been unwilling to arbitrarily block Internet communications or protocols entirely, such as WebRTC, due to the potential for significant collateral damage - Snowflake's continued operation is an example. However, this may not hold true under certain circumstances or during specific events, as demonstrated by real-world cases where platforms like YouTube or Google were blocked, such as in China.

4.4 Threat Model Countermeasures

When developing our network architecture, we implemented several countermeasures to address potential threats from State-level and Large-Scale/Continental Adversaries. These countermeasures aim to safeguard against active and passive attacks, enhance the system's resilience and ensure user anonymity and data integrity.

4.4.1 Jitter Implementation

Jitter involves deliberately altering the timing of packet transmissions within a network to mask the timing features, which can be applied within the system optionally. This strategy aims to counter active and passive network surveillance techniques such as traffic analysis and timing attacks, where adversaries exploit predictable patterns to infer sensitive information. The implementation of jitter disrupts these patterns by varying the intervals at which packets are sent, thereby complicating the task of adversaries who rely on analyzing arrival times to deduce the nature of the traffic.

We can use a distribution to achieve this variability in packet transmission timing. The Poisson distribution is apt for this application because it describes the occurrence of events that happen independently at a constant known rate, making it ideal for simulating random transmission times that do not follow any discernible pattern. Research, such as that conducted by Loopix [93], has demonstrated the effectiveness of using a Poisson distribution to implement jitter. This approach enhances the anonymity of communications, significantly reducing the risk of traffic analysis attacks.

In practice, the system generates a uniform random number u between 0 and 1 for each new packet, which is then used to calculate the transmission delay based on the Poisson (or similar) distribution. Applying this delay ensures that each packet is transmitted randomly, disrupting predictable timing patterns.

Algorithm 1 Calculate Delay for Packet Transmission.

```
1: function CalculateDelay( )
2:     u ← random()            ▷ Generate a uniform random number U between 0 and 1
3:     return ← getRndDelay(u)  ▷ Compute delay using a distribution (e.g. Poisson)
4: end function
```

4.4.2 Protocol Diversity

Utilizing a variety of communication protocols within a network infrastructure is a strategic approach to enhance security and obfuscate the nature of network traffic. This method involves the integration of multiple protocols, each with distinct characteristics and encryption standards, such as TLS (Transport Layer Security), QUIC (Quick UDP Internet Connections), and WebRTC (Web Real-Time Communication). Employing such a diverse array of protocols complicates the ability of adversaries to apply a uniform strategy to analyze or manipulate data traffic, thereby significantly enhancing the security landscape, particularly effective against active attacks, such as interception and modification, and passive attacks, like eavesdropping and traffic analysis.

4.4.3 WebRTC-based Covert Channels

Using WebRTC-based covert channels significantly enhances the complexity adversaries face when correlating traffic. This protocol supports real-time communication between browsers, which helps disguise traffic patterns crucial for countering website fingerprinting and traffic analysis. In our system, the number of packets sent via WebRTC corresponds to the number of media stream frames rather than the actual number of data packets that would typically be sent for a given user request. This makes it difficult for adversaries to recognize or match traffic patterns to known behaviours, as the system substitutes the media stream frames with packet data when handling requests.

Moreover, WebRTC introduces variability in packet timing and volume, further complicating efforts to perform traffic correlation and timing attacks. This divergence makes it challenging for adversaries to trace or infer communication details based on the observed traffic patterns, thereby enhancing user privacy and data security in potentially surveilled environments.

4.4.4 QUIC Multiplexing

Multiplexing within QUIC allows multiple independent data streams to coexist over a single connection. By interleaving multiple streams, QUIC makes it exceedingly difficult for observers to discern individual data flows or track specific user activities within the aggregated traffic. This obfuscation is a critical barrier against adversaries employing sophisticated deep packet inspection or detailed pattern analysis to extract sensitive information or infer user behaviours. Moreover, QUIC's innovative packet loss handling significantly improves over traditional protocols. Each stream within a QUIC connection

handles packet loss independently, meaning that the loss of packets in one stream does not affect the continuity or performance of others. This resilience to packet loss and corruption is particularly advantageous in mitigating the effectiveness of active attacks that rely on disrupting communication flows.

4.4.5 Packet Padding

Packet padding involves adding random data to network packets and altering their size and timing to safeguard against surveillance. This method is crucial in preventing adversaries from drawing insights based on packet sizes.

Implementing packet padding requires careful consideration to maintain network efficiency. The key is to balance the security benefits of padding with the potential increase in bandwidth usage. Ensuring that the padding is unpredictable yet not excessively burdensome on the network is crucial for maintaining optimal performance without compromising security. We will provide a more detailed explanation of how packet padding is applied in our system later in Section 4.6.3.

4.4.6 Multi-Hop Routing

Like the Tor network, multi-hop routing is a powerful technique that sends data traffic through multiple intermediaries before it reaches its final destination. This process involves layering the routes data packets take, significantly increasing the difficulty of tracing the origin or destination of the data. Distributing traffic across several nodes effectively masks the data's pathway, providing robust anonymity and security for users.

Multi-hop routing is particularly effective against sophisticated surveillance by State-Level and Large-Scope/Continental adversaries. These adversaries typically have substantial resources to monitor network communications, but multi-hop routing complicates their efforts by dispersing the data across numerous nodes spread across different geographic locations and jurisdictions. This dispersal forces adversaries to monitor multiple points along the transmission route, a logistically complex and resource-intensive task.

Furthermore, each intermediary node in a multi-hop route only knows the immediately preceding and following nodes in the chain, not the complete path of the data. This limited knowledge further secures data against potential interception by ensuring unlinkability. With a sufficient number of nodes, such as three, no single node simultaneously knows the traffic's origin and destination. As a result, no node can trace the full path of the data.

4.4.7 Packet-level Traffic Splitting

Traffic splitting involves distributing data packets across multiple independent communication channels to reduce the risk of traffic correlation attacks. By breaking up a single data stream and sending its components through different paths, the likelihood of an adversary successfully intercepting and analyzing the entire data flow is significantly

reduced. This technique effectively counters both passive and active attacks by ensuring that no single communication link carries enough information to compromise the privacy or security of the data and by masking traffic characteristics, as each path may have different latency and network conditions.

4.4.8 Multipath Multi-encapsulation

Unlike traditional traffic splitting solutions, this novel approach utilizes different encapsulation types on each path, meaning packets travelling along different routes may be encapsulated in protocols such as TLS, QUIC, or WebRTC. This multi-encapsulation strategy further strengthens privacy protections, as the varied encapsulation and network conditions across paths create unique traffic characteristics for each route. This approach counters passive and active attacks by combining packet-level splitting with diverse encapsulation methods, disrupting adversarial attempts at traffic correlation and analysis.

4.4.9 Geographic Dispersion

Although not a technique of the system itself, the deployment of its nodes should consider geographic dispersion to maximize security. Geographic dispersion works hand-in-hand with multi-hop routing, as both techniques work together to enhance the security and anonymity of network communications. While multi-hop routing ensures that data is relayed through multiple intermediary nodes to obscure the traffic's origin and destination, geographic dispersion strengthens this approach by spreading these nodes across different Autonomous System (AS) regions.

By distributing network resources and routing traffic through various geographical and administrative domains, this method effectively complicates any adversary's efforts to monitor or control the complete data flow across the network.

When network infrastructure is spread across different AS regions, it introduces redundancy and diversification crucial for performance and security. Each AS operates under distinct administrative management and potentially different jurisdictional regulations, meaning an attack or surveillance effort in one region may not necessarily affect the others. This geographical and administrative dispersion is a robust defence mechanism against large-scale monitoring or targeted attacks within a single area.

The diversity of AS regions makes it significantly more challenging for any adversary, including State-Level and Large-Scope/Continental entities, to gain comprehensive control or surveillance capabilities over the entire network. To effectively monitor or manipulate data, an adversary must establish a presence in multiple AS regions, each with its policies and protections. This requirement increases the complexity and cost of such efforts and exposes the adversary to more significant risks of detection and counteraction.

4.4.10 End-to-End Encryption

End-to-end encryption protects data by ensuring it is encrypted at the source until it reaches the intended destination, which possesses the unique decryption key. This process ensures that any intermediate nodes involved in the data transmission cannot access or decipher the plaintext of the transmitted information.

One of the primary strengths of end-to-end encryption is its ability to prevent passive and active attacks. In passive attacks, adversaries attempt to intercept and observe data, but end-to-end encryption ensures they cannot view the content. In active attacks, where adversaries may try to intercept and modify the data, encryption preserves the confidentiality of the message. While encryption's main priority is maintaining confidentiality, it also supports integrity by ensuring that data cannot be tampered with or altered without detection.

4.4.11 Summary

The diverse countermeasures integrated into our architecture ensure robust protection against various cyber threats from sophisticated adversaries. These strategies are vital in maintaining user data's confidentiality, integrity, and availability across our network. While many of the countermeasures are effective against both active and passive attacks, Table 4.1 categorizes them according to their primary objectives within the context of our system.

Active Attacks Countermeasures
End-to-End Encryption
Multi-Hop Routing
Protocol Diversity
Multipath Multi-encapsulation
Jitter
QUIC Multiplexing
Packet-level Traffic Splitting
WebRTC-based Covert Channels

Passive Attacks Countermeasures
End-to-End Encryption
Multi-Hop Routing
Jitter
Multipath Multi-encapsulation
Packet Padding
Geographic Dispersion
WebRTC-based Covert Channels
QUIC Multiplexing
Packet-level Traffic Splitting

Table 4.1: Categorization of countermeasures based on their primary objectives in addressing privacy challenges within our system, considering their applicability to both active and passive attacks.

4.5 Theoretical Protections Against Other Attack Vectors

In the preceding sections, we established a comprehensive adversary model and introduced various countermeasures designed to defend against multiple classes of adversaries with differing capabilities. Although our primary focus and evaluation have centred on protecting against website fingerprinting attacks, the same countermeasures - involving combining different protocols, adding random jitter, packet size uniformization, and

multiplexing - could, in principle, be adapted to counter a broader range of threats. This section articulates our hypotheses regarding how the proposed set of techniques may, in theory, counter various attack scenarios.

One important class of threats involves traffic correlation attacks, where adversaries attempt to link sender and receiver flows by examining timing and volume patterns across multiple network vantage points. To counteract such attempts, we hypothesize that introducing jitter disrupts predictable timing relationships: packets are delayed by random intervals, complicating efforts to link flows observed at different network locations. Packet padding ensures all transmissions appear more uniform, making size-based correlations less reliable. Employing multiple encapsulation protocols - such as TLS, QUIC, and WebRTC - can produce a heterogeneous flow profile, further impeding attempts to pair observed input and output streams. Traffic splitting under specific scenarios might also help since adversaries who can only observe part of the network will not have access to the complete traffic flow.

A related category of attacks, referred to as traffic fingerprinting, attempts to classify or infer flow characteristics based on statistical features rather than direct content inspection. We assume that traffic splitting might help mitigate this attack because, as previously stated, it might leave the adversary with less traffic to analyze. Similarly, adding jitter based on some probability distribution (e.g., uniform) may help conceal identifiable patterns and reduce the stability of any potential "fingerprint" that could be derived from timing or size patterns. Additionally, by encapsulating data within the frames of video streams (maintaining their original size and structure), WebRTC-based approaches make the protected communication appear indistinguishable from ordinary multimedia content. This seamless blending with widely used, legitimate traffic reduces distinctive statistical cues. It enhances unobservability, making it more challenging for adversaries to detect, identify, or classify the protected data. However, it is worth noting that traffic splitting with different protocols may introduce its own fingerprinting risks, which are pertinent to be studied in future work.

While these theoretical protections draw upon established principles of network obfuscation and align with findings in related research domains, their practical efficacy remains to be empirically validated. Future work will involve rigorous testing against diverse adversarial models and attack methodologies, determining how effectively these strategies, both individually and in combination, perform under real-world conditions.

4.6 System Architecture and Components

In this section, we delve into the structural intricacies of our network, focusing on two primary components: the Local Gateway Node and the Relay Node. Each type of node consists of several subcomponents, each tasked with specific functions that contribute to the overall effectiveness of the network. This detailed exploration into the architecture of

these components reveals the robust framework underpinning our network's operation, highlighting the critical role each plays in the operation of the system.

4.6.1 Local Gateway Architecture

The local gateway node has a modular architecture based on various components, depicted in Figure 4.5. Here is a brief overview of each part:

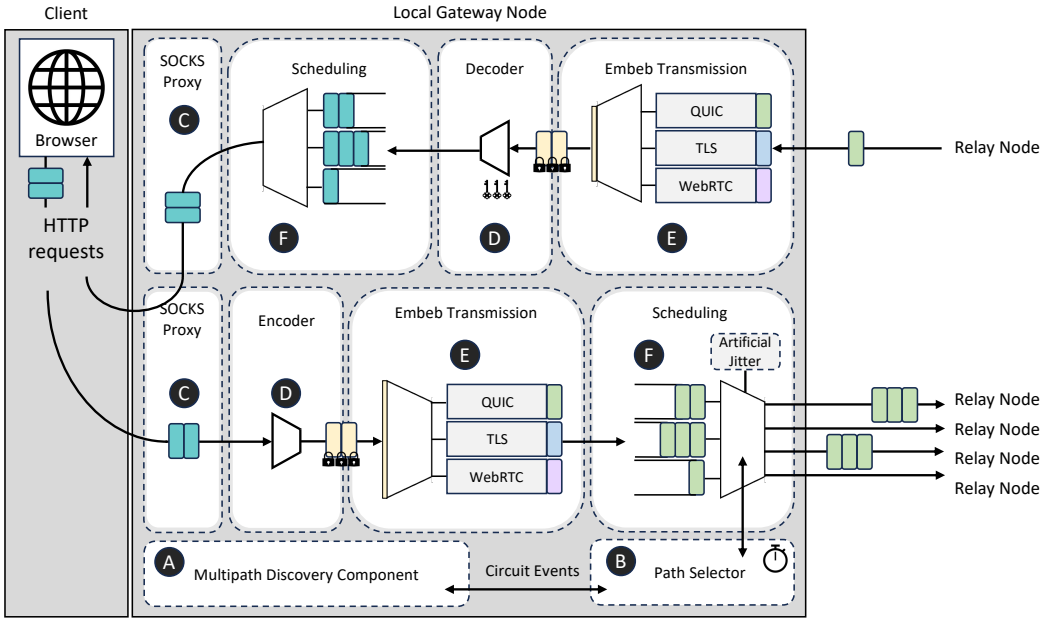


Figure 4.5: Overview of the local gateway node architecture.

Multipath Discovery Component

The Multipath Discovery Component, represented by the 'A', is responsible for identifying multiple potential paths for data transmission across the network. This component scans the available network topology to determine viable routes that can be used. Besides creating the possible paths, it uses various techniques to measure the latency (later explored in Section 4.6.6), which is particularly relevant for the chosen heuristic on the next component.

Path Selector

Once multiple paths are identified, the Path Selector component, represented by the 'B', evaluates and chooses the optimal path or paths for data transmission based on several criteria such as latency, the volume of data transmitted, time per path and potential repetition, or not, of paths. In Section 4.6.7, we provide a more detailed overview of this component and present the various algorithms for possible user-defined heuristics.

SOCKS Proxy

Now that path management has been briefly explained; we need to present how our system handles network requests from clients seeking resources from various destinations the client wishes to access. The SOCKS Proxy, represented by the 'C', is an intermediary that handles all outgoing and incoming requests via a SOCKS protocol.

There are three versions of the SOCKS protocol[1]: Version 4 (v4), Version 4a (v4a), and Version 5 (v5). Version 4 only allows the use of IP addresses as the final destination to which the proxy connects, supporting only TCP traffic and offering no authentication methods. Version 4a extends Version 4 by enabling domain names instead of just IP addresses, making it more versatile. Version 5 provides the most features, including support for multiple authentication methods, both TCP and UDP traffic, domain names, and additional enhancements like IPv6 support.

Data Encoding and Decoding

Data encoding or decoding occurs after or before, depending on the data flow (request or response, respectively) and is represented by the letter 'D' in the Figure. Regarding the encoding phase, it is responsible for the multi-layer encryption, on which the Onion Routing will later rely. In the decoding phase, it is responsible for the inverse process; this is, it takes all the layers of encryption and handles the clear data to the scheduling component. For further details, skip to the Section 4.6.3.

Embedded Transmission

Similarly, we have the embedded transmission, 'E' in the figure, that handles the integration of encoded data into various types of network traffic. The purpose of TLS, QUIC tunnels and WebRTC-based covert channels is to obscure the nature of the transmitted data, contributing further to the system's capability to provide unobservable communication channels.

Scheduling Component

Lastly, the Scheduling component ('F' in the Figure) takes charge of data transmission timing across the network. When focusing on the data forwarded from the local gateway to the relay nodes, this component strategically coordinates when and to where to send the data packets by optimizing network traffic to avoid patterns that could be detected by network surveillance. Scheduling transmissions plays a critical role in ensuring that the traffic is split across various routes, making it highly improbable that an adversary controls every route. Moreover, we can also introduce some randomness to the scheduling of the packets by introducing artificial delays to the packet's rate. When focusing on the data forwarded from the relay nodes into the local gateway, it is particularly relevant to

[1]SOCKS Protocol - https://www.rfc-editor.org/rfc/rfc1928.txt (Accessed on 29 August 2024)

mention that this component is responsible for organizing the packets by their sequence number to ensure they are sent to the SOCKS Proxy in the expected order.

4.6.2 Relay Architecture

Like the local gateway node, the relay node has a modular architecture based on various components, as depicted in Figure 4.6. Here is a brief overview of each part:

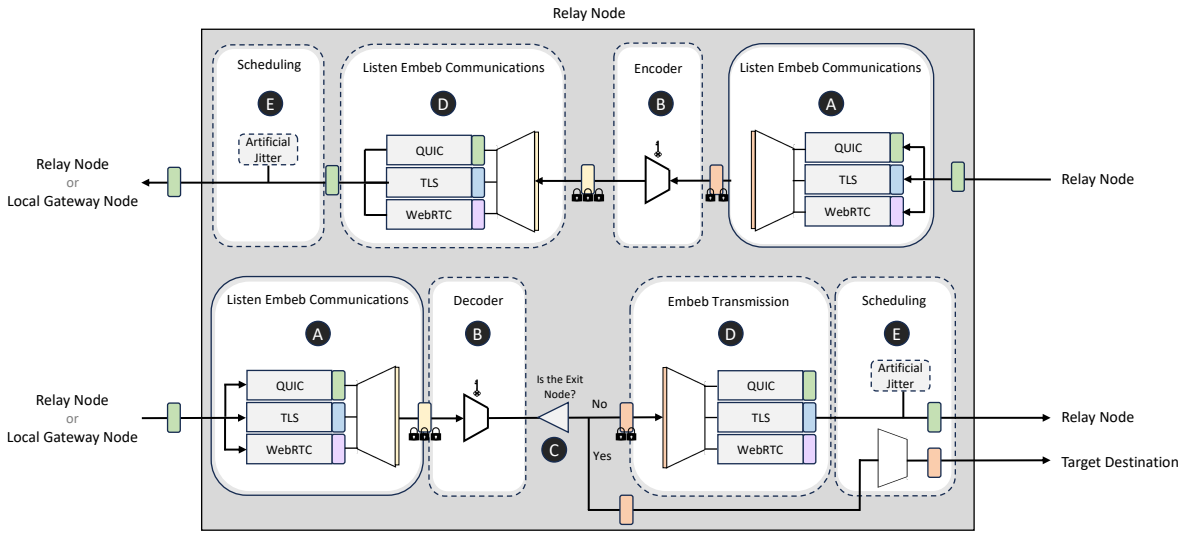


Figure 4.6: Overview of the relay node architecture.

Listen Embedded Transmissions

The Listener component, represented in the Figure by the letter 'A', listens for incoming data packets in TLS, QUIC or WebRTC communications.

Data Encoding and Decoding

Data encoding and decoding occur immediately after, referred to as 'B' in the Figure. In the encoding phase, the data is encrypted with the symmetric key the client has established with this node. While in the decoding phase, it is responsible for the inverse process; this is, it removes one layer of encryption. For further details, skip to the Section 4.6.3.

Forward Destination

If the packet is decrypted in the previous phase and contains an aggregator mark, the node will act as the aggregator for this connection. Upon confirming this role by checking the flag, the node begins functioning as the aggregator. It establishes a connection to the destination target if one does not exist; otherwise, it utilizes the existing connection. If it is not an exit node, the node decrypts the packet to retrieve the forward address and continues the pipeline to forward the packet.

Embedded Transmission

Later, we have the embedded transmission, 'D' in the Figure, that handles the integration of encoded data into various types of network traffic; if a connection is created, TLS, QUIC tunnels and WebRTC-based covert channels obscure the nature of the transmitted data, contributing further to the system's capability to provide unobservable communication channels.

Scheduling Component

Lastly, the Scheduling component ('E' in the Figure) manages data transmission across the network. It allows the introduction of random delays to avoid patterns that could be detected by network surveillance. Moreover, if it is an exit node, it schedules the packets for the destination target in their correct order, waiting for a given packet to arrive before sending one with a higher sequence number. However, if the node is not an exit node or is handling the response flow, it forwards packets directly to the next node in the path without reordering.

4.6.3 Data Encoding and Decoding

Data encoding and decoding vary greatly depending on the nature of the communication, whether it is a request or a response. In this section, we delve into the insights of both options and present the detailed structure of the actual packets. The analysis focuses specifically on communication paths with three nodes, highlighting how data is handled as it traverses through multiple intermediaries before reaching its final destination.

Request Packets

The client initiates requests by embedding specific identifiers within packets. Each packet is marked with a unique aggregator flag to inform the final node in the path that it serves as the exit node. Additionally, the packet includes a stream ID, which enables both intermediary nodes and the exit node to identify the correct decryption key associated with the user. While each user has a single stream ID, they can periodically exchange new keys and begin using them for future communications, ensuring refreshment of keys. A sequence number is added to each packet to guarantee they are properly sequenced upon reaching the exit node, ensuring they are forwarded accurately. To achieve uniformity across the network, we decided to rely on packet padding, standardizing every packet to 1024 bytes. The client then adds the data payload to the packet.

After these steps, the client encrypts the packets using AES-GCM, a robust and effective encryption standard that ensures the data's confidentiality and integrity. This encryption process introduces an additional 28 bytes of overhead to each packet, comprising a 16-byte tag and a 12-byte nonce.

Subsequently, the multilayer encryption process continues, adding the length of the forwarding address and the address itself, and the corresponding stream ID to each layer, ensuring that each node along the path can correctly identify which key to use for decryption. In Figure 4.7, we showcase the detailed structure of a packet sent to the path's entry node with 1024 bytes.

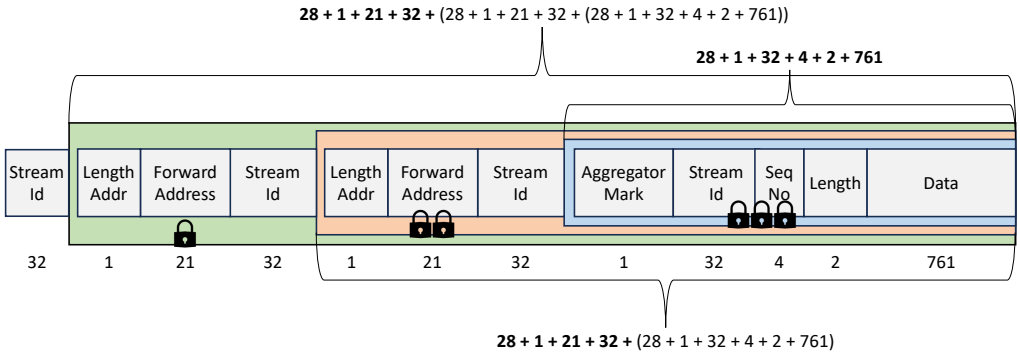


Figure 4.7: Overview of the packet structure for a request (Entry Node).

At this stage, the Relay, acting as the Entry Node, takes command. It successfully decrypts the packet and determines the next node in the path to which it must forward the remaining portion of the packet. To ensure uniformity within the network, the Relay pads the remainder of the packet to 1024 bytes before dispatching it, with this step being depicted in Figure 4.8.

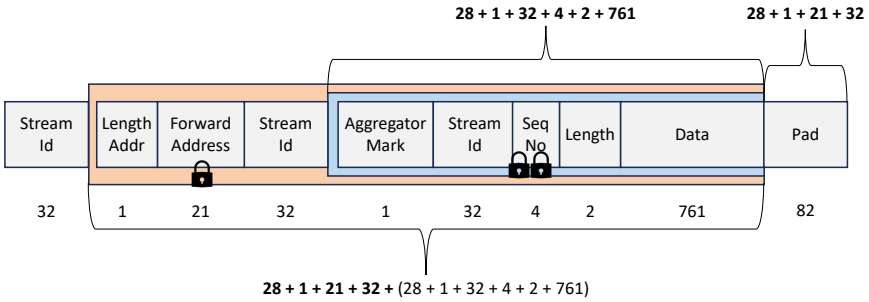


Figure 4.8: Overview of the packet structure for a request (Intermediate Node).

Following this, the intermediate node follows the documented procedure of decrypting the node, taking the length of the forward address, the respective address and the stream ID, as well as pad it, which results in a newly structured packet (with 1024 bytes), as shown in Figure 4.9, to be sent to the exit node.

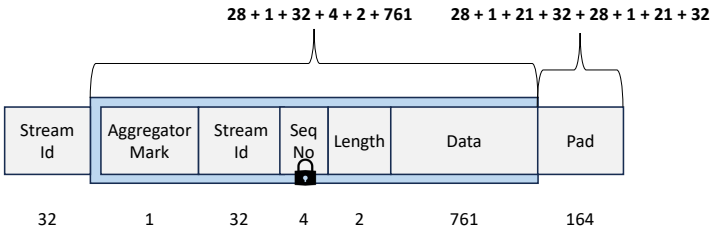


Figure 4.9: Overview of the packet structure for a request (Exit Node).

Ultimately, due to the aggregator's mark, the exit node receives the packet and realizes it is the exit node. Then, it proceeds to establish or utilize an existing connection with the designated destination and starts forwarding the incoming packets in their precise sequence. This methodical approach ensures that the packets are sent in ascending order based on their sequence numbers, thereby maintaining the integrity and order of the transmitted data.

Response Packets

The response process involves a different procedure at the exit node. Here, the exit node reads responses from the connection established with the desired destination and notes the size of the actual packet. It then encrypts the sequence number along with the data using AES-GCM. This encryption process introduces an additional 28 bytes of overhead to each packet, comprising a 16-byte tag and a 12-byte nonce. After the encryption process, the node adds padding to ensure the packet reaches the required size of 1024 bytes, as shown in Figure 4.10. After preparing the packet, the exit node sends it to the intermediate node.

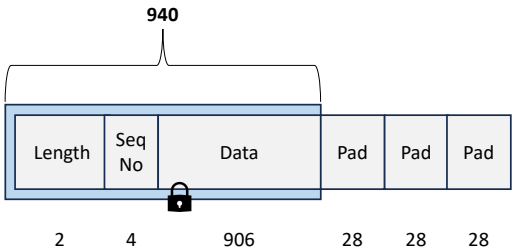


Figure 4.10: Overview of the packet structure for a response (Exit Node).

Upon receiving the packet, the intermediate node encrypts the first 968 bytes using AES-GCM, resulting in 996 bytes due to the tag and nonce. It then trims any excess to ensure the packet remains at 1024 bytes to forward it to the entry node. This packet manipulation is detailed in the structure depicted in Figure 4.11, where we can start to see the multilayer process begin to happen.

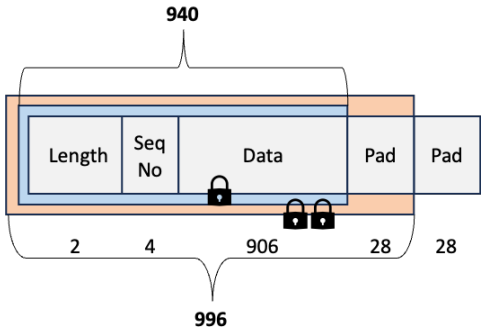


Figure 4.11: Overview of the packet structure for a response (Intermediate Node).

The entry node actively replicates the procedure executed by the intermediate node by employing AES-GCM encryption on the initial 968 bytes of the packet, resulting in 996 bytes due to the tag and nonce. It then trims the excess to ensure the packet retains a standard size of 1024 bytes. This action, detailed in Figure 4.12, prepares the packet for its final transit phase. Once the packet is ready, the entry node forwards it directly to the Local Gateway node (the client). This crucial step preserves the integrity and security of the data throughout the pipeline. Each node within this chain only knows the preceding and following nodes, ensuring unlinkability between the sender and the final destination. This approach creates a robust system where data confidentiality is tightly controlled from start to finish.

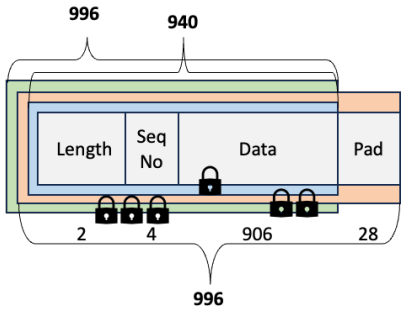


Figure 4.12: Overview of the packet structure for a response (Entry Node).

At the final stage of the transmission process, the local gateway node, equipped with all the necessary symmetric keys, decrypts each packet layer to access the clear data. After decrypting the data, the gateway node arranges and delivers it to the client application in the correct sequence.

4.6.4 Embedded Transmissions

This subsection delves into the mechanisms behind our data transmission techniques. We examine how different technologies create secure and efficient communication channels within our network architecture.

4.6.4.1 WebRTC-based Covert Channels

At the forefront of our data transmission techniques are WebRTC-based covert channels, a unique approach to secure communications. By harnessing WebRTC's real-time media capabilities, we establish covert channels that blend seamlessly with regular internet traffic, allowing data to flow discreetly.

Protocol Overview

WebRTC is built upon a series of protocols grouped into four main categories as outlined in the WebRTC protocol stack, depicted in Figure 4.13. These categories include signaling, peer connections, security measures, and communication protocols [117].

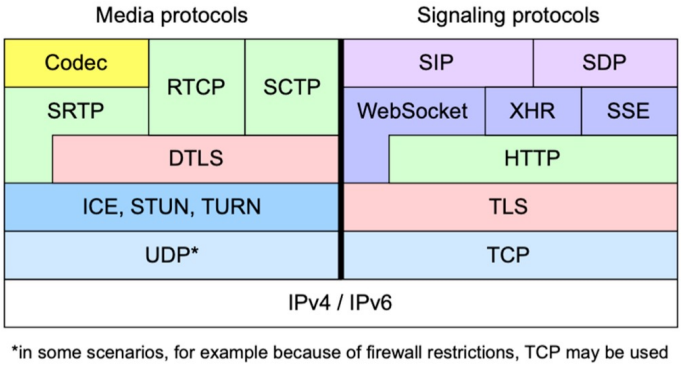


Figure 4.13: WebRTC Protocol Stack. Taken from [40].

Signaling: This stage is foundational for initiating and coordinating communication sessions between peers. It involves discovering other peers, negotiating session details, and setting up any necessary parameters for communication. While WebRTC itself does not specify a particular signaling protocol, Session Initiation Protocol (SIP) [40] is often used to facilitate these setups. The process involved in the WebRTC signaling phase is illustrated in Figure 4.14.

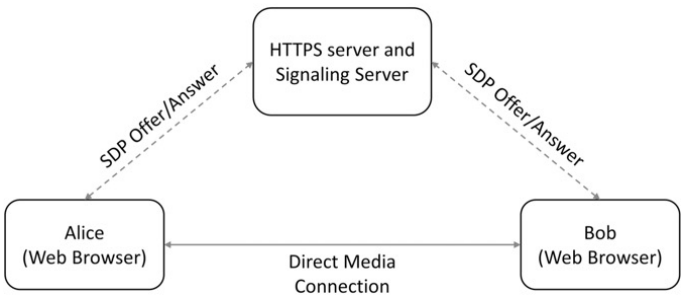


Figure 4.14: WebRTC signaling phase. Taken from [40].

Peer Connection: The Interactive Connectivity Establishment (ICE) protocol is central to setting up peer-to-peer connections. ICE handles the complexities of modern network

configurations, such as NAT traversal and firewall restrictions. It works by trying to establish a direct connection if users are on the same local network. Otherwise, it uses peer network details gathered via the STUN protocol[2]. If network constraints block direct communication, ICE[3] will then use a TURN server[4] as a relay to manage the data transfer between peers.

Security: WebRTC's security is robust; it utilizes Datagram Transport Layer Security (DTLS) for safe key exchange and Secure Real-time Transport Protocol (SRTP) to encrypt media streams. This dual-layer security ensures that all data transferred between peers is safeguarded against eavesdropping and tampering.

Communication: WebRTC relies on the Real-Time Transport Protocol (RTP) for transmitting media and the RTP Control Protocol (RTCP) for network adaptation. RTP carries the media streams, while RTCP works in the background to monitor network conditions and optimize data delivery accordingly. This adaptive mechanism is essential for maintaining the quality of service, especially in varying network environments.

Encapsulation Methodology

Our system employs an advanced method of WebRTC encapsulation, built upon the solid foundation laid by TorKameleon's [114] existing WebRTC encapsulation, particularly a new iteration of TorKameleon, reengineered in a different programming language to enhance both performance and security, which will be published on a new paper in the future.

This encapsulation technique, illustrated in Figure 4.15, actively manipulates video stream frames within our system. When a node needs to forward traffic to another machine covertly, it directly substitutes the content of video stream frames with the data of the packets to be covertly sent. This strategic substitution conceals the underlying communication, encapsulating traffic in regular video streams to evade detection. Conversely, if there is no data to forward, the node continues transmitting the video frames without alteration, maintaining the regular flow. This explains why the diagram shows an alternating pattern of normal and transformed frames, as the node only transforms the frames when there is data to forward, leaving the others unchanged. Upon reception, the receiving machine actively distinguishes between modified and standard frames, selectively disregarding the embedded data in scenarios where alteration is unnecessary, thus preserving the flow of legitimate video content. By leveraging the ubiquity of video streams, this method effectively uses video communications as a cover for covert data transfers.

[2]STUN Protocol - https://datatracker.ietf.org/doc/html/rfc5389 (Accessed on 27 September 2024)
[3]ICE Protocol - https://datatracker.ietf.org/doc/html/rfc5245 (Accessed on 27 September 2024)
[4]TURN Protocol - https://datatracker.ietf.org/doc/html/rfc5766 (Accessed on 27 September 2024)

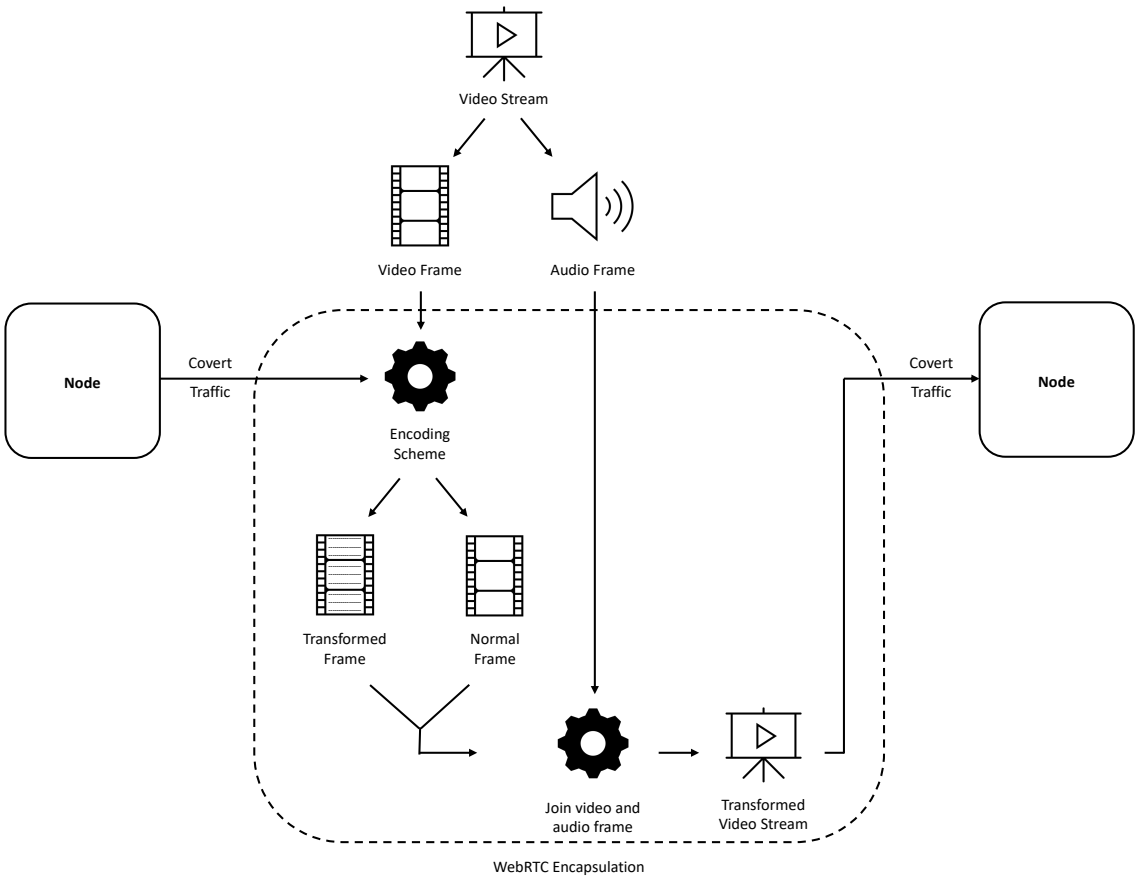


Figure 4.15: WebRTC encoding diagram.

4.6.4.2 QUIC Tunnels

QUIC (Quick UDP Internet Connections) is a modern transport layer network protocol developed primarily by Google. It operates over UDP and introduces several improvements to increase the speed and security of web communications. The protocol is structured into several key components contributing to its efficiency and robustness, and the protocol stack is depicted in Figure 4.16.

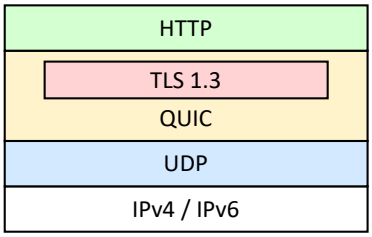


Figure 4.16: QUIC Protocol Stack.

Handshake Protocol: QUIC significantly reduces connection establishment times by combining the transport layer handshake with the cryptographic handshake, which

means that QUIC can establish a connection and negotiate encryption in a single round-trip (1-RTT), which is faster compared to the multiple exchanges required by TLS 1.1 and 1.2. Moreover, QUIC introduces the concept of 0-RTT, which allows data to be sent immediately in certain situations without waiting for a full handshake to complete. This is particularly beneficial for subsequent or repeated connections, as it allows a client to begin transmitting data immediately using previously shared cryptographic information. The 0-RTT capability minimizes latency for connections that have been previously established, leading to even faster data exchange in scenarios where low latency is crucial.

Stream Multiplexing: Unlike other known protocols relying on TCP, which are constrained by head-of-line blocking, where delays in a single stream can stall the progress of all others, QUIC enables multiple independent data streams to be transmitted concurrently over a single connection. Each stream i is managed separately, ensuring that disruptions in one stream, such as packet loss or retransmissions, do not affect the throughput or latency of others.

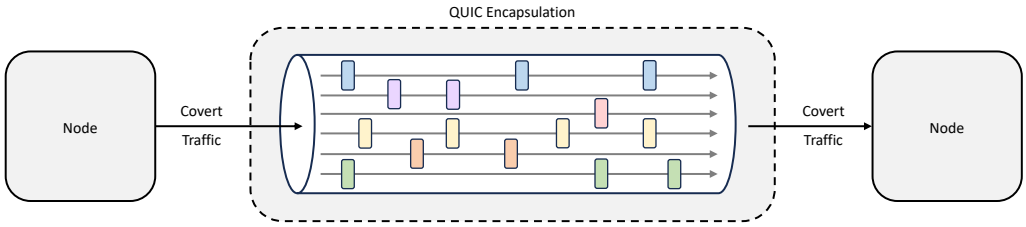


Figure 4.17: QUIC multiplexing.

Flow Control and Congestion Control: QUIC provides enhanced control over data flow through its sophisticated flow control mechanisms, which operate at both the connection and individual stream levels. This prevents any single stream from monopolizing all available resources, a feature particularly beneficial in environments with varied network capabilities. Additionally, QUIC incorporates dynamic congestion control algorithms that continuously adapt to changing network conditions in real-time. By monitoring network performance, QUIC optimizes throughput while minimizing congestion, allowing for smoother data delivery and reducing the likelihood of network bottlenecks.

Security Features: QUIC integrates security features directly into the protocol using TLS 1.3, the latest Transport Layer Security protocol version. This simplifies and enhances security implementation, providing strong end-to-end encryption. This integrated approach to encryption helps protect against several types of attacks that are possible with older protocols, such as man-in-the-middle attacks and replay attacks.

Connection Migration: A standout innovation in QUIC's design is its ability to preserve an active connection despite changes in a client's IP address. This capability is made possible through the use of Connection Identifiers (CIDs), which decouple the

connection from the IP address. Instead of relying on a fixed IP address and port pair, as traditional protocols like TCP do, QUIC assigns a unique connection ID to each session.

4.6.4.3 TLS Tunnels

TLS tunnels are critical to ensuring the privacy and integrity of data as it traverses our network. By encapsulating data within TLS, we safeguard it against interception and tampering. This is particularly vital when sensitive information must be transmitted securely over potentially insecure networks. Our implementation of TLS tunnels emphasizes robust encryption algorithms and up-to-date security protocols to maintain the highest standards of data protection.

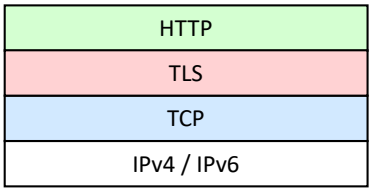


Figure 4.18: TLS Protocol Stack.

Robust Security: TLS (Transport Layer Security) protocols secure data transmission through asymmetric cryptography for key exchange, symmetric encryption for data privacy, and hash functions for ensuring message integrity. Older versions like TLS 1.0 and TLS 1.1 are being phased out due to security vulnerabilities, while TLS 1.2 and TLS 1.3 offer stronger protections. TLS 1.3, in particular, enhances security and performance by simplifying the handshake process and eliminating outdated algorithms, ensuring that data remains secure against evolving threats.

Handshake Protocol: The TLS handshake is a vital process where a client and server exchange messages to establish a secure connection. This involves negotiating cryptographic algorithms, exchanging encryption keys, and authenticating both parties. In TLS 1.2, this process requires multiple round trips between the client and server to complete the handshake. However, TLS 1.3 reduced the number of steps required in the handshake, which results in faster connection establishment while also improving security by eliminating outdated cryptographic algorithms.

Forward Secrecy: A crucial security feature in TLS 1.2 and later versions, designed to ensure that session keys remain secure even if the server's private key is compromised in the future. By using ephemeral key exchange mechanisms, such as Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman (ECDHE), each session generates unique, temporary keys that are discarded after use. This guarantees that even if an attacker gains access to the server's private key, they cannot decrypt past communications, providing an additional

layer of protection against retrospective attacks.

TLS Multiplexing: Unlike QUIC, which natively supports the multiplexing of multiple data streams within a single connection, TLS traditionally handles multiplexing at a higher layer. In TLS, multiplexing typically involves managing multiple independent connections, each encapsulated within its own secure TLS session, as shown in Figure 4.19. Additionally, individual packets from different incoming connections can be aggregated into a single data stream for transmission over the secure TLS tunnel. Once transmitted, these packets are decrypted and redirected to their respective destinations, as depicted in Figure 4.20.

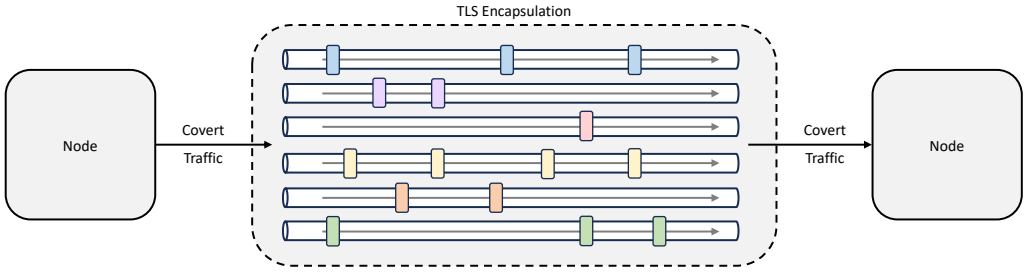


Figure 4.19: TLS multiplexing in various connections.

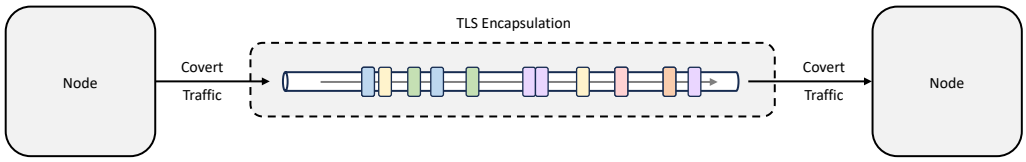


Figure 4.20: TLS multiplexing in one connection.

4.6.5 Key-Exchange Protocol

In onion routing, each data packet is wrapped in multiple layers of encryption, with each layer being decrypted as the packet passes through successive nodes in the network circuit. This design requires the local gateway (or client) to establish a unique symmetric key with each relay the data traverses. In order to ensure data security and privacy, these symmetric keys must be distinct for each client.

We designed a dynamic key exchange protocol rather than relying on a static method, where symmetric keys must be pre-established and managed independently of the routing process. In this protocol, the client holds the public keys of the relays and leverages asymmetric encryption to negotiate a unique symmetric key with each relay securely. This approach ensures that keys are securely generated and exchanged at the start of the communication session, eliminating the need for pre-configured symmetric keys and significantly improving flexibility and security. One potential method for obtaining the public keys of the relays is through a directory server, similar to the approach used in the Tor network.

Our system goes a step further in ensuring security by supporting the periodic renewal of the key exchange process. This feature allows fresh symmetric keys to be generated between the client and relay over time, thereby mitigating risks associated with long-term key reuse.

This process starts with the local gateway node creating a symmetric key and associating it with a stream ID, as explained in Section 4.6.3. This data is then encrypted using the public key of the entry node. Once encrypted, the packet is sent to the entry node, initiating the secure communication, as illustrated in Figure 4.21. This process allows only the entry node, which holds the corresponding private key, to decrypt and access the packet's contents, ensuring that the communication is secure right from the outset, depicted in Figure 4.22.

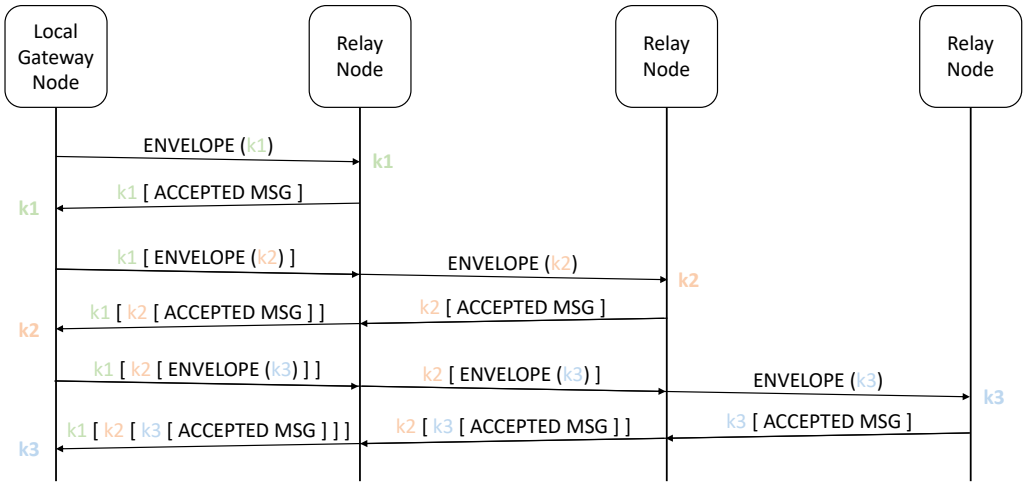


Figure 4.21: Key Exchange Protocol.

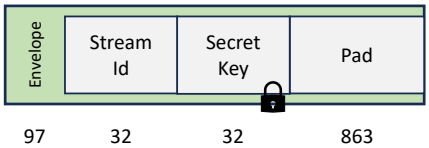


Figure 4.22: Packet for the Entry Node Key Exchange.

Following successful decryption, the entry node utilizes the symmetric key it just acquired to encrypt and send a response back to the local gateway node, as shown in Figure 4.23. This response serves as a confirmation that the entry node was able to decrypt the packet and is ready to communicate securely using the symmetric key. Upon receiving this response, the local gateway node advances to the next phase of establishing secure paths through the network.

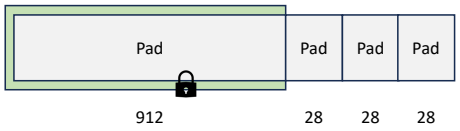


Figure 4.23: Response Packet for the Entry Node Key Exchange.

In this next phase, the local gateway node prepares a new packet for the intermediate node. This packet includes another symmetric key and the intermediate node's address. The packet is encrypted first with the intermediate node's public key, ensuring that only this node can decrypt it, and then with the symmetric key provided to the entry node, depicted in Figure 4.24. After receiving this doubly encrypted packet, the entry node removes the first encryption layer using its symmetric key and forwards the packet to the intermediate node.

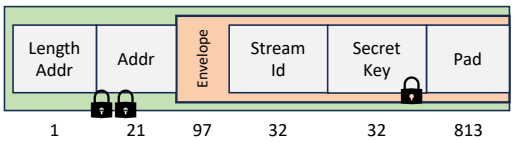


Figure 4.24: Packet for the Intermediate Node Key Exchange.

The intermediate node decrypts the packet using its private key, extracts the symmetric key, and sends a response back encrypted with this new key and later on encrypted with the entry's node key as well, as shown in Figure 4.25. This confirms that the intermediate node is also ready for secure communication. The local gateway node now prepares for the final link in the chain - the exit node.

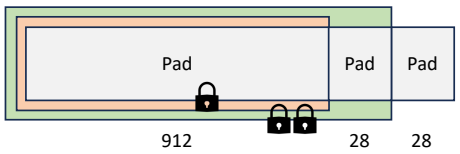


Figure 4.25: Response Packet for the Intermediate Node Key Exchange.

For the exit node, the local gateway creates a packet encrypted with the exit node's public key. This packet is further encrypted with the symmetric keys of the intermediate and entry nodes consecutively, adding multiple layers of security, as illustrated in Figure 4.26. Each node along the route strips off its corresponding layer of encryption, with the exit node finally using its private key to decrypt the last layer.

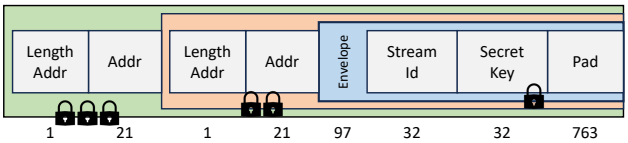


Figure 4.26: Packet for the Exit Node Key Exchange.

Once the exit node decrypts its layer, it sends a response back using the symmetric key it just acquired to encrypt the message, represented in Figure 4.27. This encrypted message then travels back through the network, getting re-encrypted by each node's symmetric key. When the local gateway node finally receives this message, it decrypts each layer, confirming successful and secure communication across all nodes involved in the routing process.

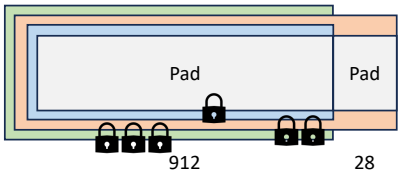


Figure 4.27: Response Packet for the Exit Node Key Exchange.

4.6.6 Latency Measurement

In Onion Routing, where data traverses multiple network nodes, the focus is on ensuring privacy and security, with latency being a secondary consideration, particularly in systems like Tor. However, accurate latency measurement may help optimize performance and improve user experience. This exploration focuses on three advanced latency measurement techniques - geolocation measurement, end-to-end measurement, and ping measurement - designed to support heuristic-based latency estimation. These heuristics will be used for path selection in the system, though other heuristic approaches are also considered.

4.6.6.1 Geolocation measurement: Theoretical Latency Estimation

Geolocation measurement is a theoretical approach to latency estimation, leveraging the geographic coordinates - longitude and latitude - of each node within the network. By calculating the distance between nodes, this method declares that latency is proportionally linked to the distance traversed. This estimation can be achieved with the help of an API request, such as http://ip-api.com/json, which retrieves the necessary geolocation data of the nodes involved. However, if this information is already available in the same location where public keys are stored, such as in a directory server like in the case of Tor, it would streamline the process. Although this approach does not consider the physical realities of cable routing and other infrastructural elements that may influence transmission times, it provides a baseline estimation that can be particularly valuable in scenarios where network resources are limited. This method avoids the need for extensive communication with multiple nodes, conserving bandwidth and reducing overhead by not having to evaluate every potential end-to-end latency scenario empirically. The pseudocode for this approach is demonstrated in Algorithm 2.

Algorithm 2 Latency Measurement based on Geographic Locations

```
1: function HANDLELATENCYMEASUREMENT (lat, lon)
2:     clientGeopoint ← newGeoPoint(lat, lon)
3:     numPath ← 0
4:     while numPath < length(paths) do
5:         relays ← paths[numPath]
6:         locations ← []
7:         latency ← 0
8:         for i ← 0 to length(destinationProps) − 1 do
9:             auxGeopoint ← NewGeoPoint(relays[i].Lat, relays[i].Lon)
10:            locations ← locations ∪ auxGeopoint
11:            if i > 0 then
12:                latency ← latency + distance(locations[i − 1], locations[i])
13:            else
14:                latency ← latency + distance(clientGeopoint, locations[i])
15:            end if
16:        end for
17:        latencies ← latencies ∪ {ConnID : numPath, Lat : latency}
18:        numPath ← numPath + 1
19:    end while
20: end function
```

4.6.6.2 End-to-end measurement: Practical Latency Evaluation

Moving from theoretical to empirical, end-to-end measurement offers a comprehensive assessment of latency by tracking the round-trip time of data packets as they travel from the source node to the exit node and back. This journey through the network's encrypted layers provides a realistic measure of latency, capturing delays induced by various factors, including node processing times, network congestion, and the encryption/decryption processes inherent to our solution. The practical data from this method allows us to evaluate the actual performance of different paths through the network, making it an invaluable tool for selecting the paths with the best latencies. The pseudocode for this approach is shown in Algorithm 3.

4.6.6.3 Ping measurement: Real-time Latency Diagnostics

To complement theoretical and empirical methods, ping measurement provides a real-time diagnostic tool that assesses network latency through active probing. By sending ping requests to selected nodes and measuring the time it takes for responses to return, this method not only offers immediate feedback on network health but also aids in the creation of a latency matrix. This matrix is instrumental in analyzing the latency across different nodes and routes within the network, thus enabling a detailed assessment of network performance and the identification of potential bottlenecks. The pseudocode for this approach is depicted in Algorithm 4.

Algorithm 3 Latency Measurement based on End-to-End Time

```
1:  function handleLatencyMeasurement ( )
2:      numPath ← 0
3:      while numPath < length(paths) do
4:          relays ← paths[numPath]
5:          tlsConn, err ← tls.Connection("tcp", relays[0].Addr)
6:          if err == nil then
7:              startTime ← time.Now()
8:              tlsConn.write(latencyMeasurementPacket)
9:              new Thread(handleLatencyResponses(tlsConn, numPath, startTime))
10:         end if
11:         numPath ← numPath + 1
12:     end while
13: end function
14:
15: function handleLatencyResponses (tlsConn, numPath, startTime)
16:     buf ← make([]byte, 1024)
17:     err ← io.ReadFull(tlsConn, buf)
18:     if err == nil then
19:         endTime ← time.Now()
20:         latency ← endTime − startTime
21:         latencies ← latencies ∪ {ConnID : numPath, Lat : latency}
22:         tlsConn.Close()
23:     end if
24: end function
```

4.6.7 Path Selection

When managing data transfer in networked environments, selecting the most effective path for routing data is crucial for performance and ensuring robust security. Path selection strategies cater to different operational requirements and challenges, emphasizing efficiency, fairness, and optimal resource utilization. Each strategy has its own advantages and potential drawbacks, making the choice highly context-dependent.

In our system, we first create the possible path combinations from the available nodes, and then we apply the path selection strategies to choose the best path for the data transfer. Currently, we employ a randomized approach fro selecting paths with different protocols (e.g., TLS, QUIC, or WebRTC). In the future, additional heuristics could be implemented, enabling more tailored selections based on factors like latency, load, or protocol-specific security features. Below, we delve deeper into these strategies, exploring how they work and their specific benefits within our system.

4.6.7.1 Round-Robin Path Selection

This method provides a systematic approach to distribute network traffic evenly across all available paths. Cycling through each path in a sequential and fair order ensures that no single path is overwhelmed by excessive load. Distributing traffic across all possible paths

Algorithm 4 Latency Measurement based on Ping Requests

```
1: function handleLatencyMeasurement (clientAddr, selectedNodes)
2:     listenLatencyResults(clientAddr, selectedNodes)
3:     myLatencyResults(clientAddr, selectedNodes)
4: end function
5:
6: function listenLatencyResults (cAddr, selectedNodes)
7:     numNodes ← 0
8:     while numNodes < length(selectedNodes) do
9:         node ← selectedNodes[numNodes]
10:        tlsConn, err ← tls.Connection("tcp", node.Addr)
11:        if err == nil then
12:            tlsConn.write(selectedNodes)
13:            new Thread(handleLatencyResponses(tlsConn, node.Addr))
14:        end if
15:        numNodes ← numNodes + 1
16:    end while
17: end function
18:
19: function handleLatencyResponses (tlsConn, nodeAddr)
20:    latencyResults, err ← tlsConn.read()
21:    if err == nil then
22:        latencies[nodeAddr] ← latencies[nodeAddr] ∪ latencyResults
23:        tlsConn.Close()
24:    end if
25: end function
26:
27: function myLatencyResults (cAddr, selectedNodes)
28:    numNodes ← 0
29:    while numNodes < length(selectedNodes) do
30:        node ← selectedNodes[numNodes]
31:        latency ← ping(node.Addr)
32:        latencies[cAddr] ← latencies[cAddr] ∪ {Dest : node.Addr, Lat : latency}
33:        numNodes ← numNodes + 1
34:    end while
35: end function
```

without repetition complicates potential eavesdropping or interference by adversaries, as they are unlikely to control every node within the network.

Algorithm 5 Round-Robin Path Selection

```
1: function pathSelection (latencies, usedPaths, usedNodes)
2:     i ← 0
3:     while i < length(latencies) && (usedPaths[latencies[i].ConnID] ||
       (!intersect(usedNodes, paths[latencies[i].ConnID]) || !hasKeyExchanged(i))) do
4:         i ← i + 1
5:     end while
6:     if i < length(latencies) then
7:         usedPaths[latencies[i].ConnID] ← true
8:         usedNodes ← usedNodes ∪ paths[latencies[i].ConnID]
9:         currentPath ← paths[latencies[i].ConnID]
10:    else
11:        i ← 0
12:        usedPaths ← {}
13:        usedNodes ← []
14:    end if
15: end function
```

4.6.7.2 Random Path Selection

In contrast to the predictable nature of Round-Robin, Random Path Selection introduces a layer of unpredictability, which can be particularly beneficial in dynamic network environments where path conditions may change rapidly. This method randomly selects from available paths for each data transfer, potentially choosing the same path multiple times. While this can lead to uneven path utilization and the risk of temporary overloads, it significantly increases the difficulty for any external entities attempting to predict or track the data flow.

Algorithm 6 Random Path Selection

```
1: function pathSelection (latencies)
2:     randomIndex ← getRandomIndex(latencies)
3:     currentPath ← paths[latencies[randomIndex].ConnID]
4: end function
```

4.6.7.3 Weighted Unused and Used Random Path Selection

Building on the essential random selection, the Weighted Unused and Used Random Path Selection strategy introduces a refined approach by categorizing paths into 'used' and 'unused' based on their selection in the current cycle. This strategy uses a weighted random mechanism where unused paths - those not yet chosen in the ongoing cycle - are given higher probabilities. This promotes a more equitable distribution of network load and helps prevent any single path from becoming a bottleneck.

Algorithm 7 Weighted Unused and Used Random Path Selection

```
1: function pathSelection (latencies, usedPaths, usedNodes, threshold)
2:     i ← 0
3:     usedIndices ← []
4:     unusedIndices ← []
5:     while i < length(latencies) do
6:         if usedPaths[latencies[i].ConnID] then
7:             usedIndices ← usedIndices ∪ {i}
8:         else if (!intersect(usedNodes, paths[latencies[i].ConnID]) ||
   hasKeyExchanged(i)) then
9:             unusedIndices ← unusedIndices ∪ {i}
10:        end if
11:        i ← i + 1
12:    end while
13:    if length(unusedIndices) > 0 then
14:        randomWeight ← random()
15:        if randomWeight < threshold then
16:            randomIndex ← random(unusedIndices)
17:            usedPaths[latencies[randomIndex].ConnID] ← true
18:            usedNodes ← usedNodes ∪ paths[latencies[randomIndex].ConnID]
19:        else
20:            randomIndex ← random(usedIndices)
21:        end if
22:        currentPath ← paths[latencies[randomIndex].ConnID]
23:    else
24:        usedPaths ← {}
25:        usedNodes ← []
26:    end if
27: end function
```

4.6.7.4 Weighted Random Path Selection

Finally, this selection leverages performance metrics to influence path choice. This method calculates weights based on the quality of paths - paths with better performance metrics (lower latency) receive higher weights. During selection, a cumulative distribution function is built based on these weights, and a random number is generated to select a path probabilistically favouring higher-weighted paths.

Additionally, an alternative weighted random path selection approach that we did not follow - discussed in the CoMPS framework - considers generating weights using a non-uniform probability distribution for each new connection. This technique leverages the Dirichlet distribution to introduce greater flexibility, allowing paths to be selected based on a dynamic probability distribution. We could easily add this approach to our system, enabling more adaptive path selection based on connection-specific probabilities.

Algorithm 8 Weighted Random Path Selection

```
1:  function pathSelection (latencies)
2:      i ← 0
3:      totalWeight ← 0
4:      while i < length(latencies) do
5:          totalWeight ← totalWeight + 1/(i + 1)
6:          i ← i + 1
7:      end while
8:      randomWeight ← random(totalWeight)
9:      i ← 0
10:     weightSum ← 0
11:     while i < length(latencies) do
12:         weightSum ← weightSum + 1/(i + 1)
13:         if weightSum > randomWeight then
14:             break
15:         end if
16:         i ← i + 1
17:     end while
18:     currentPath ← paths[latencies[i].ConnID]
19: end function
```

4.6.8 Change Path Signal

In MIRACE, dynamic path switching is vital for breaking traffic patterns. When the system detects that the current path needs to change through the heuristics-defined time or is packet-based, it reroutes packets through a newly selected path. However, simply forwarding packets via the new path is insufficient to maintain effective traffic splitting.

Irregular traffic flow, such as when a user is downloading a file and no new packets are sent for an extended period after the path switch, can lead to the system continuing to route responses through the old path. This disrupts the traffic-splitting strategy, harming the system's ability to distribute traffic across multiple routes and compromising security and performance.

A special packet is transmitted immediately after a path change to address this. This packet notifies the aggregator that all subsequent responses must be routed through the newly established path. Without this signal, the old path could inadvertently remain in use, risking exposure and diminishing the system's ability to stay resilient against censorship.

To implement the signal, our design leverages the Data Encoding outlined in Section 4.6.3. The special packet is marked with a unique identifier that the aggregator recognizes, allowing it to interpret the packet as a signal to switch to a new path. This new path corresponds to the message's route, enabling the aggregator to reroute subsequent packets along this updated route seamlessly. Due to the encryption inherent in onion routing, the remaining nodes in the network are unaware of the upcoming path change, ensuring that only the aggregator knows the routing update and guaranteeing the routing

of subsequent responses through the updated route.

This mechanism is instrumental in upholding MIRACE's core objectives of security, privacy, and censorship resistance, even during periods of unidirectional traffic, such as extended file downloads or any single-direction data transmission.

4.7 MIRACE Protocol Stack

Finally, in Figure 4.28, we present the designed MIRACE protocol stack, which illustrates the system's design. This stack encompasses three sophisticated forms of encapsulation: WebRTC, QUIC, and TLS.

The WebRTC-based encapsulation uses RTP on DTLS to embed covert data into the frames of the video. This encapsulation closely follows the standard WebRTC protocol stack but with the addition of an application layer protocol designed specifically to embed covert data directly into the RTP stream.

The QUIC-based encapsulation allows for secure and efficient multiplexing of data streams within a single connection. It operates using its own built-in encryption and transport mechanisms, similar to the traditional QUIC protocol stack but includes a specialized application layer that facilitates the embedding of covert data within the QUIC stream itself, ensuring that hidden data can be transmitted.

The TLS-based encapsulation provides end-to-end encryption over secure communication channels. Like standard TLS stacks, it ensures confidentiality and integrity by encrypting all transmitted data. However, in this implementation, an application layer protocol is added to allow the embedding of covert data within the encrypted TLS packets, ensuring that covert communication is securely wrapped within the TLS encryption layer.

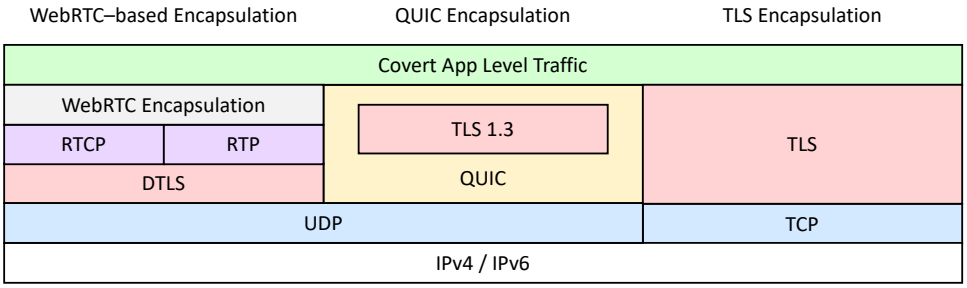


Figure 4.28: MIRACE protocol stack.

4.8 Summary

This chapter introduced MIRACE, a censorship-resistant tunnelling tool to enhance online anonymity and privacy. It outlined the system's adversary model, addressing both passive and active threats, and focused on three core objectives: resistance to website fingerprinting attacks, unblockability, and maintaining reasonable performance. To achieve these goals, MIRACE uses dynamic multipath routing at the packet level, incorporates

diverse protocols like TLS, QUIC, and WebRTC for data encapsulation, and applies Onion Routing for multiple layers of encryption.

Several countermeasures were discussed to mitigate the identified threats. These include jitter, which adds randomness to packet timing, data tunnelling and encapsulation (i.e., QUIC and TLS tunnelling and WebRTC media-based covert channels) to obscure traffic patterns, and traffic padding to prevent adversaries from gaining insights based on packet sizes. Protocol diversity, traffic splitting, and packet padding further complicate adversaries' efforts to analyze traffic. Additionally, multi-hop routing and geographic dispersion ensure that data traverses multiple intermediary nodes, guaranteeing unlinkability between the user and the accessed content. End-to-end encryption remains critical for safeguarding both privacy and data integrity.

The chapter also explored MIRACE's architecture, detailing the roles of the local gateway, relay nodes, and critical components such as data encoding and decoding, key exchange protocols, and latency measurement. Additionally, the chapter also provided information on the heuristics available for both path selection, allowing users to optimize the system based on performance or security needs, giving them greater flexibility and control over its operation.

In conclusion, this chapter demonstrated how MIRACE addresses some of the challenges of censorship resistance and how it delivers privacy protections while maintaining reasonable performance.

5

Implementation and Prototype

In this chapter, we explore the design, development, and technical implementation of the MIRACE system. The chapter provides a comprehensive overview of the system's architecture, modular components, and the technology stack used. It emphasizes the design principles adopted, the specific choices in communication protocols, and the methods employed to ensure system flexibility, scalability, and adaptability. Moreover, we detail the testing environment used to validate the system's performance under different scenarios, offering insight into the complexities encountered and how they were resolved in the implementation process.

5.1 Prototype Overview

The MIRACE prototype is designed with a modular architecture emphasizing flexibility, adaptability, and ease of maintenance. At its core, the system revolves around two primary node types: the Local Gateway Node and the Relay Node, forming the network's backbone.

One of the standout features of MIRACE is its support for multiple covert communication channels, including WebRTC, QUIC, and TLS. The WebRTC encapsulation is based on a Golang version of TorKameleon [114], which will soon be published. To make that integration, we had to change the flow of communications to allow for chain communication and integrate it with the predefined packet structure explained in Section 4.6.3 and all the processes of route management. Then, regarding the QUIC and TLS encapsulation, we developed them from scratch with the help of some well-known packages from the Golang library.

Path selection and routing within MIRACE offer significant flexibility. Users can choose from various strategies, such as round-robin, random, weighted random on latency results, or based on path utilization. Path changes can be triggered by different factors, including time intervals, the number of packets sent, or the total amount of packets communicated.

Additionally, the system allows users to define the number of communication hops between nodes, providing further control over how data travels through the network. MIRACE can either multiplex client connections into a single stream or handle each one

separately, depending on the specific requirements of the deployment.

Latency calculations are also highly customizable, as MIRACE supports various metrics, including geolocation, end-to-end performance, and ping results. This modular approach to latency ensures that users can tailor the system to meet their performance needs.

In order to ensure seamless deployment across multiple environments, the entire system was containerized using Docker 24.0.7. Although development and testing were conducted on macOS 14.5, the containerized nature of MIRACE ensures it can be easily deployed on other platforms with minimal configuration.

The modular design of the prototype ensures that each component, whether it is a communication protocol or a latency calculation method, operates independently. This separation allows for individual development, testing, and optimization, ensuring that any component can be swapped out or removed without impacting the entire system. This structure also enhances scalability, enabling MIRACE to adapt to different network environments and operational demands.

5.2 Technology

The development of the MIRACE system involved using several Go libraries and external packages to handle different aspects of communication and functionality, as presented in Table 5.1. At its core, Go's standard libraries, such as `errors` for error handling, `context` for managing API contexts, and `net` for network operations, provided essential features for system operations. These packages allowed the system to efficiently handle basic tasks like I/O, network communication, and synchronization.

The `crypto/tls` package was used to implement TLS 1.2 and 1.3 for secure communication, ensuring encrypted data transmission across the network. This provided the necessary foundation for handling secure connections where needed.

The system also used the `quic-go` library to implement the QUIC protocol, enabling communication over UDP. This protocol functioned independently, supporting additional communication capabilities within MIRACE.

WebRTC was integrated using the `pion/webrtc/v3` package to support real-time peer-to-peer communication. This allowed for direct node communication, with the `pion/rtp` package handling the Real-time Transport Protocol for data packetization.

Additional packages were employed to support other functionalities. The `gopkg.in/yaml.v2` library was used for configuration management, allowing easy setup and adjustments, while `go.uber.org/atomic` provided tools for safe concurrent operations. The `go-shadowsocks2` library enabled SOCKS proxy support, and `gorilla/websocket` was used for signaling in the context of TorKameleon's WebRTC encapsulation method.

Together, these libraries and tools formed the technological part of MIRACE, ensuring it could handle a variety of communication methods and perform efficiently across different environments.

Packages	Description
"errors"	Package errors implements functions to manipulate errors.
"context"	Package context defines the context type,which has values across API boundaries and between processes.
"os"	Package os provides a platform-independent interface to operating system functionality.
"net"	Package net provides a interface for network I/O, including TCP/IP, UDP, domain name resolution, and sockets.
"io"	Package io provides basic interfaces to I/O primitives.
"fmt"	Package fmt implements formatted I/O with functions analogous to C's printf and scanf.
"bytes"	Package bytes implements functions for the manipulation of byte slices.
"crypto"	Package crypto collects common cryptographic constants.
"time"	Package time provides functionality for measuring and displaying time.
"math"	Package math provides basic constants and mathematical functions.
"sort"	Package sort provides primitives for sorting slices and user-defined collections.
"strings"	Package strings implements simple functions to manipulate UTF-8 encoded strings.
"sync"	Package sync provides basic synchronization primitives such as mutual exclusion locks.
"encoding"	Package encoding defines interfaces that convert data to and from byte-level and textual representations.
"flag"	Package flag implements command-line flag parsing.
"gopkg.in/yaml.v2"	Package yaml implements YAML support for the Go language.
"go.uber.org/atomic"	Package atomic provides simple wrappers around numerics to enforce atomic access.
"github.com/shadowsocks/go-shadowsocks2"	Package socks implements essential parts of SOCKS protocol.
"github.com/quic-go/quic-go"	Package quic-go is an implementation of the QUIC protocol (RFC 9000, RFC 9001, RFC 9002) in Go.
"crypto/tls"	Package tls partially implements TLS 1.2, as specified in RFC 5246, and TLS 1.3, as specified in RFC 8446.
"github.com/gorilla/websocket"	Package websocket implements the WebSocket protocol defined in RFC 6455.
"github.com/pion/rtp"	Package rtp provides RTP packetizer and depacketizer.
"github.com/pion/webrtc/v3"	Package webrtc implements the WebRTC 1.0 as defined in W3C WebRTC specification document.

Table 5.1: Packages used in MIRACE System.

5.3 Complexity

Our system's architecture is delineated by two types of nodes, each represented by distinct packages within our codebase. These packages not only define the core functionalities of each node type but also encapsulate the intricacies necessary for their specialized operations.

Below, in Table 5.2, we present each package's lines of code (LoC) and the amount of disk space they occupy. This visualization aids in understanding the scale and the allocation of resources within our system, ensuring clarity for maintenance and scalability assessments.

It is important to note that, regarding our WebRTC implementation, the displayed size of each package does not include the size of the video used to cover the traffic, as the implementation is independent of the video used and can greatly vary in size. Thus, their exclusion from the size metrics provides a more focused view of the static components of our system.

To provide a clearer understanding of our system, we present visual representations of a gateway and a proxy, which correspond to the first two packages detailed in Table 5.2. Figures 5.1 and 5.2 illustrate the modules of these packages along with their respective sizes, which use TLS encapsulation and measure latency based on geolocation.

In Table 5.3, we present the complexity metrics of the system, as extracted through

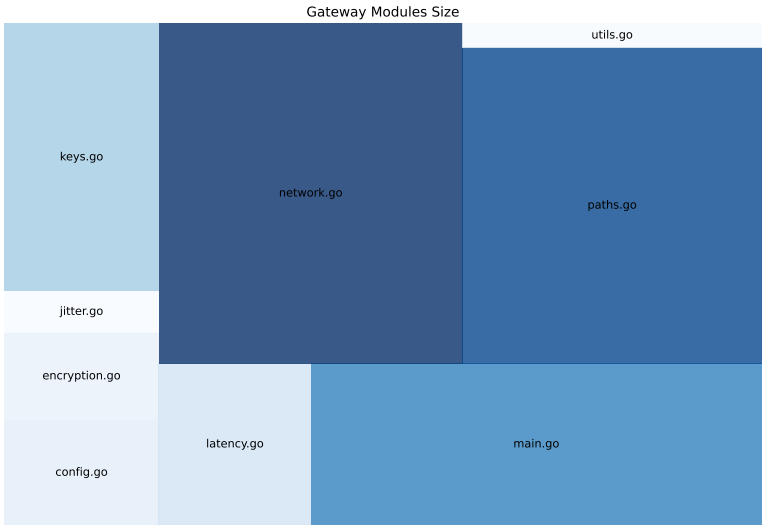


Figure 5.1: Module size breakdown in a Gateway Node using TLS encapsulation and measuring latency based on geolocation.

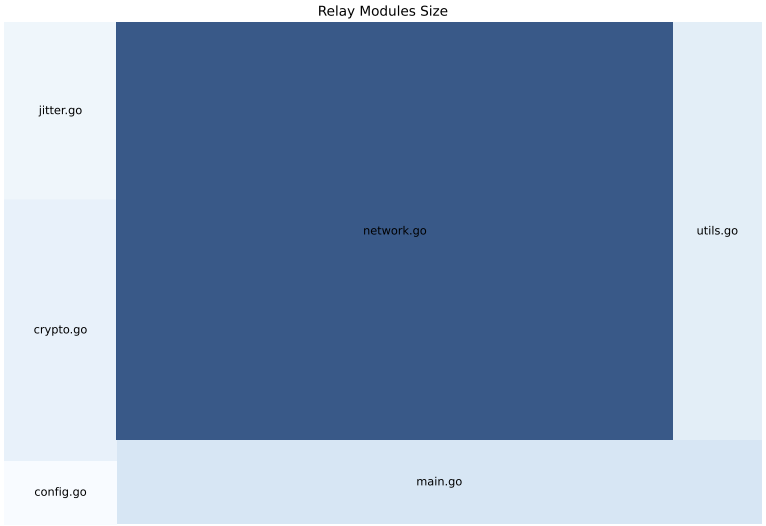


Figure 5.2: Module size breakdown in a Relay Node using TLS encapsulation and measuring latency based on geolocation.

Package	Lines of Code	Size of Package
socks5-tls/with-onion-routing/different-connection/latency-calc/client	1733	169 KB
socks5-tls/with-onion-routing/different-connection/latency-calc/proxy	800	142 KB
socks5-tls/with-onion-routing/different-connection/latency-end-to-end/client	1784	170 KB
socks5-tls/with-onion-routing/different-connection/latency-end-to-end/proxy	823	143 KB
socks5-tls/with-onion-routing/different-connection/latency-ping/client	1887	173 KB
socks5-tls/with-onion-routing/different-connection/latency-ping/proxy	970	147 KB
socks5-tls/with-onion-routing/same-connection/latency-calc/client	1807	170 KB
socks5-tls/with-onion-routing/same-connection/latency-calc/proxy	857	144 KB
socks5-tls/with-onion-routing/same-connection/latency-end-to-end/client	1859	171 KB
socks5-tls/with-onion-routing/same-connection/latency-end-to-end/proxy	858	144 KB
socks5-tls/with-onion-routing/same-connection/latency-ping/client	1957	175 KB
socks5-tls/with-onion-routing/same-connection/latency-ping/proxy	1025	148 KB
socks5-quic/with-onion-routing/different-connection/latency-calc/client	1747	170 KB
socks5-quic/with-onion-routing/different-connection/latency-calc/proxy	793	144 KB
socks5-quic/with-onion-routing/different-connection/latency-end-to-end/client	1811	172 KB
socks5-quic/with-onion-routing/different-connection/latency-end-to-end/proxy	818	144 KB
socks5-quic/with-onion-routing/different-connection/latency-ping/client	1913	175 KB
socks5-quic/with-onion-routing/different-connection/latency-ping/proxy	993	149 KB
socks5-quic/with-onion-routing/same-connection/latency-calc/client	1797	172 KB
socks5-quic/with-onion-routing/same-connection/latency-calc/proxy	845	145 KB
socks5-quic/with-onion-routing/same-connection/latency-end-to-end/client	1858	174 KB
socks5-quic/with-onion-routing/same-connection/latency-end-to-end/proxy	870	146 KB
socks5-quic/with-onion-routing/same-connection/latency-ping/client	1964	177 KB
socks5-quic/with-onion-routing/same-connection/latency-ping/proxy	1039	150 KB
socks5-kameleon/with-onion-routing/packets/latency-calc/key-exchange-quic--webrtc-per-addr/client	2434	195 KB
socks5-kameleon/with-onion-routing/packets/latency-calc/key-exchange-quic--webrtc-per-addr/proxy	2472	194 KB
socks5-kameleon/with-onion-routing/packets/latency-calc/key-exchange-quic--webrtc-per-conn/client	2433	195 KB
socks5-kameleon/with-onion-routing/packets/latency-calc/key-exchange-quic--webrtc-per-conn/proxy	2475	194 KB
socks5-kameleon/with-onion-routing/packets/latency-end-to-end/key-exchange-and-latency-quic--webrtc-per-addr/client	2558	198 KB
socks5-kameleon/with-onion-routing/packets/latency-end-to-end/key-exchange-and-latency-quic--webrtc-per-addr/proxy	2501	194 KB
socks5-kameleon/with-onion-routing/packets/latency-end-to-end/key-exchange-and-latency-quic--webrtc-per-conn/client	2561	198 KB
socks5-kameleon/with-onion-routing/packets/latency-end-to-end/key-exchange-and-latency-quic--webrtc-per-conn/proxy	2500	194 KB
socks5-kameleon/with-onion-routing/packets/latency-ping/key-exchange-and-latency-quic--webrtc-per-addr/client	2582	200 KB
socks5-kameleon/with-onion-routing/packets/latency-ping/key-exchange-and-latency-quic--webrtc-per-addr/proxy	2649	198 KB
socks5-kameleon/with-onion-routing/packets/latency-ping/key-exchange-and-latency-quic--webrtc-per-conn/client	2580	200 KB
socks5-kameleon/with-onion-routing/packets/latency-ping/key-exchange-and-latency-quic--webrtc-per-conn/proxy	2649	199 KB
All directories	**63202**	**6173 KB (6.02 MB)**

Table 5.2: Lines of code and size of each package of the MIRACE System.

Go Report Card [99]. Go Report Card is a widely used tool for evaluating Go codebases, offering analysis across multiple key metrics to ensure that code adheres to best practices. It plays a valuable role by automating the review process, allowing developers to maintain consistency and compliance with Go standards while also highlighting areas for improvement in code quality.

Several tools are integrated into the Go Report Card to assist this evaluation. Each project, corresponding to a specific encapsulation type (TLS, QUIC, and WebRTC), was analyzed individually. Tools such as `gofmt` for code formatting, `go_vet` for static analysis, `ineffassign` for detecting inefficient variable assignments, and `misspell` for catching spelling errors, all reported 100% compliance across the system's encapsulation types (TLS, QUIC, and WebRTC). This high level of compliance demonstrates a strong adherence to Go's coding standards, ensuring that the code is well-formatted, free from common errors, optimized for efficiency, and devoid of spelling mistakes.

Directory	TLS	QUIC	WebRTC	All (Total Project)
Files	92	92	110	**294**
gofmt	100%	100%	100%	**100%**
go_vet	100%	100%	100%	**100%**
ineffassign	100%	100%	100%	**100%**
license	0%	0%	0%	**100%**
gocyclo	90%	86%	67%	**80%**
misspell	100%	100%	100%	**100%**
Grade	**A+ 93.0%**	**A+ 92.6%**	**A+ 90.3%**	**A+ 97.7%**

Table 5.3: Code complexity metrics extracted using Go Report Card.

The `gocyclo` metric, which measures cyclomatic complexity, is another crucial aspect of the Go Report Card. Go Report Card recommends keeping the complexity score below 15, as lower scores generally indicate simpler and more maintainable code. While the system shows well-managed complexity with an 80% score, the WebRTC component scored slightly lower at 67%, likely due to the inherent complexity involved in the complex process of encapsulating traffic within video streams in the WebRTC module.

Despite this slight drop in the WebRTC component, the overall scores from the Go Report Card are excellent, reflecting high-quality code and diligent maintenance across the project. The TLS component scored 93.0%, QUIC 92.6%, and WebRTC 90.3%, all contributing to an overall grade of A+ for the system.

5.4 Instalation and Setup

Follow the steps below to install and run the MIRACE system on your local machine.

1. **Prerequisites**
   Ensure you have the following installed:

   - **Go** version 1.21.5 or later.

2. **Clone the Repository**
   First, clone the repository to your local machine by running the following command:

   ```
   git clone https://github.com/hugogspereira/mirace.git
   ```

3. **Navigate to the Project Directory**
   Change into the project directory using the following command:

   ```
   cd mirace
   ```

4. **Choose Protocol and Latency Calculation Method**
   Select the desired protocol (WebRTC, QUIC, or TLS) and latency calculation method

(geolocation, end-to-end performance, or ping results) by navigating to the appropriate directory:

- **For SOCKS5 into QUIC:**

```
cd socks5-quic/latency-?????
```

- **For SOCKS5 into TLS:**

```
cd socks5-tls/latency-?????
```

- **For SOCKS5 into WebRTC:**

```
cd socks5-kameleon/latency-?????
```

Replace `latency-?????` with the appropriate latency calculation method (`latency-calc`, `latency-end-to-end`, or `latency-ping`).

5. **Run the Client Component**
To start the client component, execute one of the following commands:

- **For Time Interval Path Changes:**

```
./gateway -listenPort=????? -excludedRegions=CountryName1
   ,CountryName2,etc -changePathEvery=XhYmZs \
    -strategy=rr/rnd/w-rnd/w-rnd-unused
```

- **For Packet Count Path Changes (Sent Packets):**

```
./gateway -listenPort=????? -excludedRegions=CountryName1
   ,CountryName2,etc -changePathEveryXSentPackets=X \
    -strategy=rr/rnd/w-rnd/w-rnd-unused
```

- **For Packet Count Path Changes (Total Packets):**

```
./gateway -listenPort=????? -excludedRegions=CountryName1
   ,CountryName2,etc -changePathEveryXTotalPackets=X \
    -strategy=rr/rnd/w-rnd/w-rnd-unused
```

Replace the placeholders with your specific configuration:

- `?????`: the desired listen port.

- `CountryName1,CountryName2,etc`: the regions to exclude.
- `rr/rnd/w-rnd/w-rnd-unused`: the desired path selection heuristic.
- `XhYmZs`: the desired time interval for path change.
- `X`: the desired packet count for path change.

6. **Run the Proxy Component**
   Finally, run the proxy component by executing the following command:

   ```
   ./relay -listenPort=?????
   ```

   Replace `?????` with the desired listen port.

Ensure you replace all placeholders with appropriate values based on your specific setup requirements.

5.5 Deployment for Validation and Testing

Before deploying the MIRACE system across multiple VPSs worldwide, we rigorously validated its functionality and performance in a local environment. Testing was conducted in both native setups and Docker containers to simulate various deployment scenarios, and automated scripts were developed to streamline testing, ensuring consistent and efficient validation. So, for users looking to quickly run MIRACE with minimal configurations, a quick-start option is available. Simply execute the following script to launch the system without needing extensive setup:

Note: Ensure that SOCKS5 is properly configured on port 1080 and running on the client machine before proceeding with the following steps.

Running Locally

1. Navigate to the `tests` folder:

   ```
   cd mirace/tests
   ```

2. Run the provided script to launch one gateway and nine relays locally:

   ```
   sh run_locally_one_client_multiple_relays.sh
   ```

Running with Docker

1. Navigate to the `tests` folder:

   ```
   cd mirace/tests
   ```

2. Use the provided script to build and run the Docker containers (one gateway and nine relays):

```
sh build_and_run_docker_containers.sh
```

5.6 Summary

In this chapter, we detailed the design, development, and implementation of the MIRACE system, emphasizing its modular architecture and flexibility. We explored the various communication protocols supported by the system, including WebRTC, QUIC, and TLS, and highlighted the role of Golang packages in building a robust and scalable system.

We also discussed the complexity of the system's architecture, breaking down the lines of code and disk space occupied by each package. The Go Report Card analysis further reinforced the quality of the code, indicating strong adherence to coding standards across the system.

Furthermore, we provided detailed instructions for installing and setting up the MIRACE system, including options for running the system locally or within Docker containers. The validation process, which involved rigorous testing across multiple environments, ensured the system was reliable, scalable, and ready for deployment in distributed multi-server environments.

6

Experimental Evaluation

When evaluating network systems, it is imperative to establish comprehensive validation metrics and assessment methodologies. These methods serve a dual purpose: they assess the system's efficiency and performance and ensure that it adheres to the requisites of the threat model. This section examines the critical metrics employed in system validation, including resistance to website fingerprinting attacks, user-perceived performance, and resource allocation.

6.1 Methodology

In this section, we outline the evaluation methodology, starting with the objectives of the experimental evaluation. Following this, we discuss the test environment setup and provide details on the tools utilized during testing, the critical metrics observed, and our approach to testing and validation.

6.1.1 Evaluation Goals

Our experimental evaluation focused primarily on four key aspects: performance testing, resource usage by the system, and resistance to website fingerprinting. The performance tests enabled us to evaluate the developed tool in real-world scenarios, providing specific results for metrics commonly used to assess network systems, such as throughput and latency.

Aside from performance, we also delved into practical use cases for the system, evaluating the performance it offers in these scenarios and the feasibility of its real-world deployment. Furthermore, we examined the system's impact on computational resources, particularly CPU and memory allocation during regular operations.

Finally, we conducted fingerprinting resistance tests to evaluate the tool's ability to withstand attacks targeting website fingerprinting, such as those carried out by adversaries using traffic analysis techniques. This provided insight into how well the system can defend against modern fingerprinting detection models, especially those leveraging machine learning. Metrics such as accuracy, precision, recall, F1-score, Matthews Correlation

Coefficient, Cohen's Kappa Score, Log Loss, Jaccard Score and F2-Score were measured to assess the tool's resistance and robustness against website fingerprinting attacks.

6.1.2 Test Bench Environment

The testing environment for our setup was designed to ensure both robustness and geographical diversity. We employed nine OVH Virtual Private Servers (VPS) as Relay Nodes, each configured to run Ubuntu 24.04, the operating system's latest stable release, during our tests. These servers were provisioned with modest hardware specifications: 4 virtual CPU cores, each operating at a clock speed of 2.0 GHz, alongside 8 GB of RAM. While these specifications were relatively modest, they were adequate for the scope of our testing, allowing us to simulate typical workloads without overprovisioning resources. Additionally, each VPS was equipped with a 1 Gbps network interface, ensuring sufficient connectivity for efficient data transmission.

The geographical distribution of the Relay Nodes was a critical aspect of our setup, enabling us to simulate a globally distributed network environment. The servers were strategically located across multiple continents, with nodes in France, Germany, the United Kingdom, Poland, Canada, Australia, and Singapore. This distribution allowed us to test the network's performance under varying latencies and conditions, reflecting real-world scenarios.

For the Gateway Node, we utilized a personal machine based in Portugal. This machine was an Apple M2 Pro, featuring a 12-core CPU and 16 GB of RAM, providing robust computational capabilities to handle the demands of the gateway role in our network.

6.1.3 Video Specifications for WebRTC Test Environment

For our WebRTC encapsulation evaluation, we used a video stream to simulate real-world media traffic within the system. The video chosen for this test was a 4K (3840 x 2160 pixels) clip encoded using the VP8 codec. It consisted of 2805 frames, running at 30 frames per second (FPS), with a header size of 32 bytes. This resulted in a video duration of approximately 1 minute and 33.5 seconds.

This video's high resolution and frame rate created a challenging and realistic environment for evaluating the system's performance. It allowed us to thoroughly assess how effectively the system handled encapsulation, packet manipulation, and traffic management under the load of a high-definition media stream. Additionally, larger video frame sizes provide more capacity for embedding traffic, making accommodating more data within each frame easier and potentially increasing throughput by minimizing the need to split data packets across multiple frames.

To better understand the distribution and variability of frame sizes throughout the video, we compare frame size per frame number in Figure 6.1. This is essential for evaluating how much traffic can be embedded within each frame, as fluctuations in frame size can affect the efficiency of data encapsulation and overall system throughput. It is

important to note that the first frame of the video, which has a size of 330433 bytes, is not included in the plot. The size of this frame deviates markedly from the others due to its inclusion of crucial initialization data and metadata required for video playback, and by excluding this outlier allows a clearer analysis of the regular variability in frame sizes.

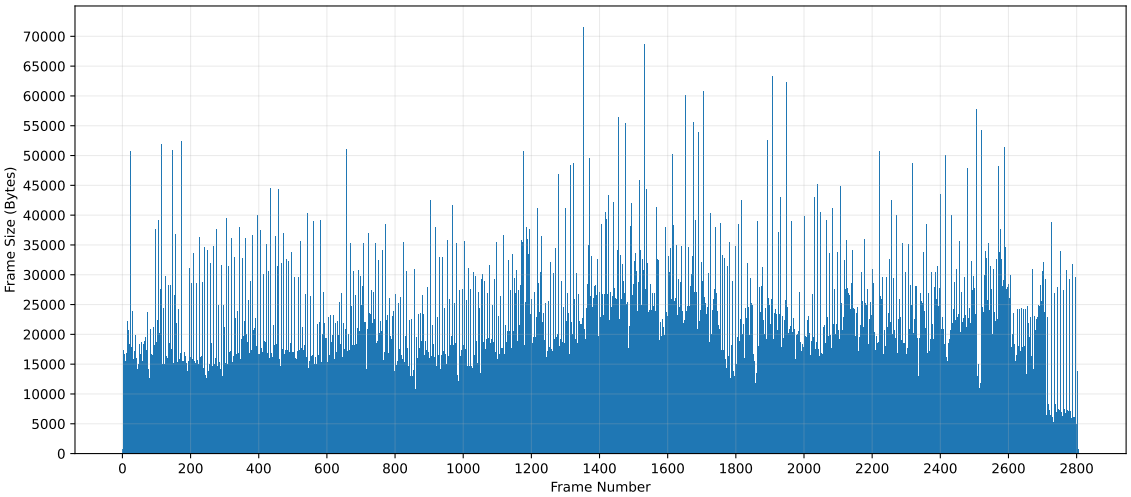


Figure 6.1: Comparison of frame sizes across the video stream.

6.2 Performance

The performance perceived by the system's end users is a crucial factor in determining a network system's usability and overall quality. This metric typically correlates with the system's responsiveness, encompassing latency, download speeds, and the overall user experience. To accurately gauge this performance, we adopted a methodology akin to the one used by the Tor Network. This involved download tests files[1] of 1MB from an HTTP Server located in the United Kingdom. This approach enabled us to measure two critical performance metrics: throughput, determined by the download speed, and latency, defined as the time elapsed from the initiation of the download request until the first byte of the response was received.

Our evaluation encompassed various network configurations, focusing mainly on different traffic encapsulation techniques and the number of relays involved in forming the circuits while keeping the circuit fixed. We employed an algorithm for the traffic splitting mechanism that switched circuits based on the number of packets sent and received. Then, we evaluate it using various methods (round-robin, random, weighted random based on latency and on unused or used paths) to select the subsequent circuit from the pool of established circuits. We tested this traffic-splitting configuration with different numbers of circuits available in the user's pool, applying the various encapsulation methodologies.

[1] Download Test Files - https://www.thinkbroadband.com/download (Accessed on 3 August 2024)

6.2.1 Number of Nodes Per Circuit: Throughput and Latency

In this first test, we aimed to understand the effect of the number of relays in a circuit on throughput and latency across all types of traffic encapsulation for a single client. Our evaluation demonstrated that throughput ranged from 2.12 Mbps (using the WebRTC-based encapsulation methodology with different-connections configuration - where each stream establishes its independent connection - and circuits of 4 relays) to 46.45 Mbps (using the QUIC encapsulation methodology with the same-connection configuration - where the path shares a single continuous connection that multiplexes multiple client streams - and circuits of 1 relay), as depicted in Figure 6.2. Throughput appeared reasonable across all configurations for common Internet activities, even with circuits of 3 relays encapsulating traffic within WebRTC video frames between all relays. We also present the results for latency in Figure 6.3. This metric inherently includes the overhead associated with establishing connections for all evaluated encapsulation strategies: TLS, QUIC, and WebRTC. However, for the same-connection configuration, the latency impact is only noticeable on the first request, as subsequent requests will use an already-established connection. Additionally, it is notable to mention that each data point presented in the graphs represents the average result of ten iterations of the same test, providing a reliable performance measure under each configuration.

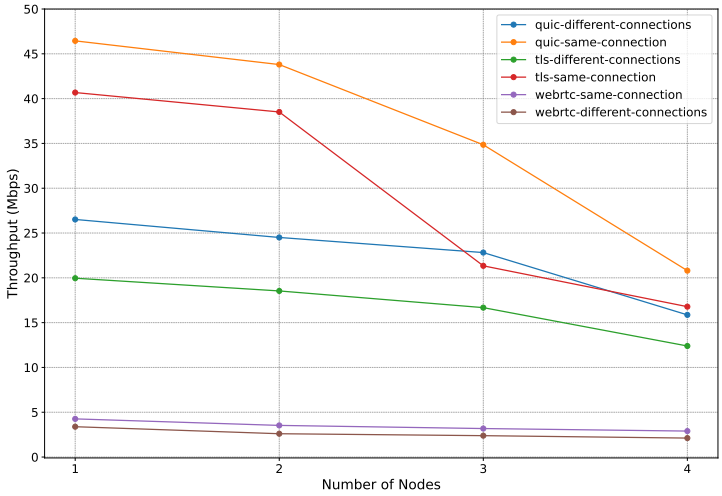


Figure 6.2: Throughput results for all encapsulation strategies across different circuit sizes.

Some conclusions can be drawn from these results when comparing encapsulation strategies with the same configuration (i.e., same-connection and different-connections). First, as expected, the performance of the WebRTC-based encapsulation strategy is lower than that of the others. This is due to the limited number of video frames with fixed sizes available for embedding traffic, which can vary significantly depending on the video used as the carrier. QUIC generally shows the highest performance, as it is a more recent optimized protocol, while TLS also performs reasonably well. Also, when using the same-connection configuration, the overhead of establishing new connections

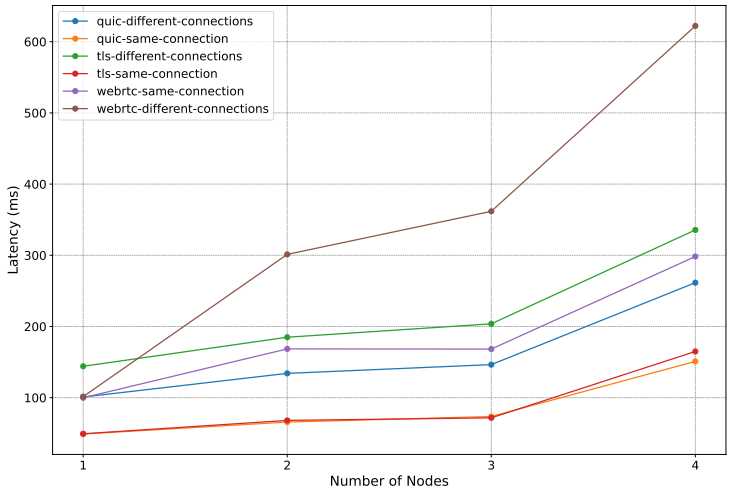


Figure 6.3: Latency results for all encapsulation strategies across different circuit sizes.

is less pronounced than the different-connections configuration, leading to lower latency in the former. The latency for WebRTC-based encapsulation in the different-connections configuration is particularly high due to the additional requirements for establishing peer-to-peer WebRTC connections between all relays in the circuit (i.e., the signalling protocol).

6.2.2 Bootstrap Time: Key-Exchange Protocol

In this section, we evaluate the bootstrap time required for establishing circuits, focusing specifically on circuits with three relays, similar to those used in Tor. The bootstrap time refers to the duration necessary to exchange cryptographic keys between the relays using the protocol presented in Section 4.6.5.

Our measurements, as summarized in Table 6.1, show the bootstrap times for exchanging keys across a path of 3 nodes located in France, the UK, and Germany, respectively. Each value presented in the table represents the average of over 25 trials, ensuring the robustness and reliability of the results.

Protocol	Number of Key	Time (ms)	Country
quic-same-connection	1	45.95	France
quic-same-connection	2	60.14	UK
quic-same-connection	3	69.06	Germany

Table 6.1: Bootstrap time measurements for circuits with three nodes using QUIC and the same-connection configuration.

The results indicate that the bootstrap time increases as more hops are involved, and also each additional key negotiation introduces an extra layer of encryption. For example, with one key, the circuit only negotiates and operates with a single encryption layer, resulting in the fastest bootstrap time of 45.95 milliseconds. When introducing a second key, the circuit manages two encryption layers, and consequently, the message has to cross

one more hop, leading to an increased bootstrap time of 60.14 milliseconds. Similarly, with three keys, the circuit handles three layers of encryption, further increasing the bootstrap time to 69.06 milliseconds. Despite the incremental increases in bootstrap time as more keys and encryption layers are added, the overall time remains reasonably low.

While these measurements are specific to the QUIC protocol, we assume similar behaviour would be observed with other protocols, such as TLS or WebRTC. The differences between these protocols will likely result in variations in bootstrap time on the order of tens or even a few hundred milliseconds. However, we believe such differences will not significantly impact the overall insights derived from this assessment, allowing us to generalize the findings across the various secure communication methods.

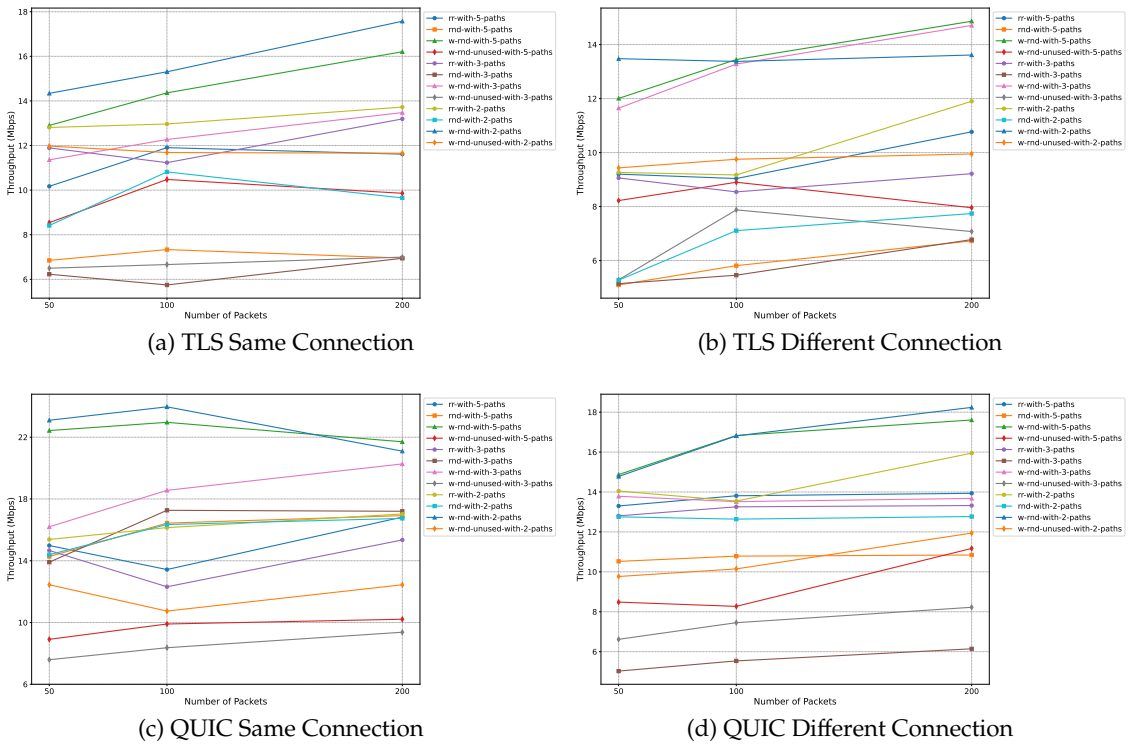
6.2.3 Traffic Splitting: Throughput and Latency

To further investigate throughput and latency over various paths, we implemented and assessed several traffic-splitting heuristics: round robin (rr), random path selection (rnd), weighted-random based on latency (w-rnd), and weighted-random based on unused paths (w-rnd-unused). Circuit selection, performed in the aforementioned manners, was triggered based on the total number of packets sent and received, with thresholds set at 50, 100, and 200 packets.

Additionally, we varied the number of available circuits, testing configurations with 2, 3 and 5 circuits in the user's pool. Inspired by the default Tor circuit size, each circuit was configured with 3 relay nodes for this test. The results for throughput and latency are presented in Figures 6.4 and 6.5, respectively, where each data point presented in the graphs represents the average result of ten iterations of the same test, providing a reliable performance measure under each configuration.

For the WebRTC-based encapsulation methodology derived from TorKameleon, the reliance on UDP and the lack of mechanisms for packet loss handling and retransmission made subsequent tests challenging to assess accurately, especially when multiple hops employ WebRTC, as each hop introduces potential points of failure. This issue is particularly problematic for data integrity in file transfers, where missing packets result in corrupted files and lead to test failures. However, this limitation poses less of a concern for applications that inherently support reliability, such as web browsing. Some performance impacts of using this WebRTC-based encapsulation are further discussed in the TorKameleon paper [114]. We are also anticipating a new version, reckoning on Turbo Tunnel [39], which aims to address these challenges and improve the reliability of our assessments.

Our analysis of the throughput and latency across different path selection heuristics reveals several key findings. The weighted-random (w-rnd) heuristic based on latency consistently delivered the highest throughput and the lowest latency across both TLS and QUIC protocols. This result aligns with expectations, as the heuristic prioritizes paths with the lowest latency, leading to more efficient data transmission. In contrast,



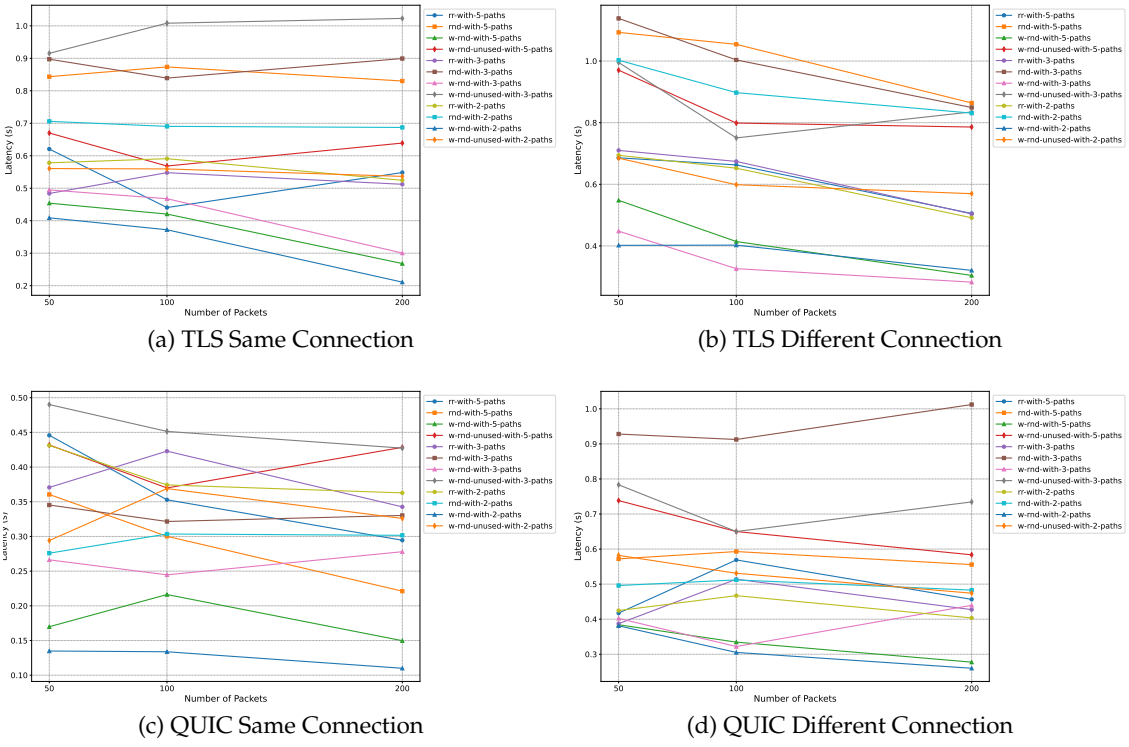
(a) TLS Same Connection

(b) TLS Different Connection

(c) QUIC Same Connection

(d) QUIC Different Connection

Figure 6.4: Throughput Variations with Different Packet Count for Path Changes Across Diverse Protocols and Heuristics.



(a) TLS Same Connection

(b) TLS Different Connection

(c) QUIC Same Connection

(d) QUIC Different Connection

Figure 6.5: Latency Variations with Different Packet Count for Path Changes Across Diverse Protocols and Heuristics.

the random (rnd) and weighted-random-unused (w-rnd-unused) heuristics showed the lowest throughput and the highest latency, as they do not take latency or other performance metrics into account when selecting paths, resulting in less optimal data flow.

When examining the effect of packet numbers (50, 100, and 200 packets), we observed that, in most cases, throughput increased with larger packet numbers while latency decreased. However, no clear trend emerged across all configurations, indicating that the impact of packet size on performance metrics is not uniform.

One of the most notable findings was the impact of the number of available paths on performance. The two-path configuration often outperformed setups with three or five paths. This can be attributed to the geographical distribution of relays in our testing environment. When using fewer paths, the selection of geographically closer relays was more likely, resulting in better performance. By contrast, configurations with more paths sometimes involved routing data through more distant relays, increasing latency and decreasing throughput due to the greater complexity and overhead involved in managing multiple paths.

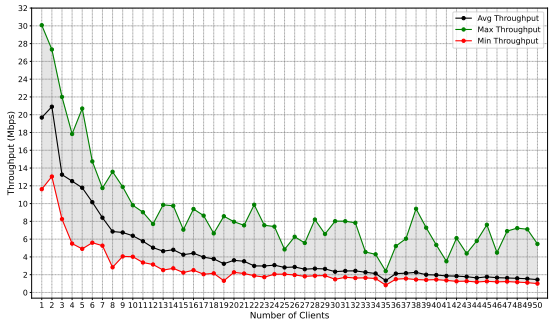
Regarding the protocol used, QUIC generally outperformed TLS in terms of overall throughput and latency. This is consistent with previous research highlighting the efficiency and performance benefits of QUIC over traditional protocols like TLS. The results underscore the importance of selecting the appropriate protocol for secure data transmission, particularly in scenarios where performance is a critical factor.

6.2.4 Scalability: Throughput and Latency

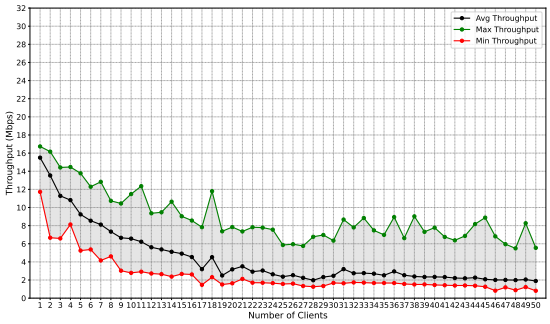
To assess the scalability of our system, we conducted a thorough performance evaluation focusing on throughput and latency across a client range from 1 to 50, over one circuit. The results for throughput, illustrated in Figure 6.6, demonstrate the system's capability to handle increasing client loads while maintaining efficient data transfer rates. Concurrently, the latency measurements, presented in Figure 6.7, provide insights into the responsiveness of the system under varying client loads. Each data point present in the aforementioned figures represents an average of over 25 values, ensuring reliable performance measurements.

When examining throughput, QUIC consistently outperforms TLS, particularly under lighter client loads. In the same connection configuration, QUIC proves more efficient, delivering higher throughput rates than TLS. However, as the client load increases, the performance gap between the two protocols narrows, exhibiting similar throughput degradation at higher loads. This convergence suggests that while QUIC is optimized for handling smaller loads more effectively, its advantage diminishes as the number of clients scales.

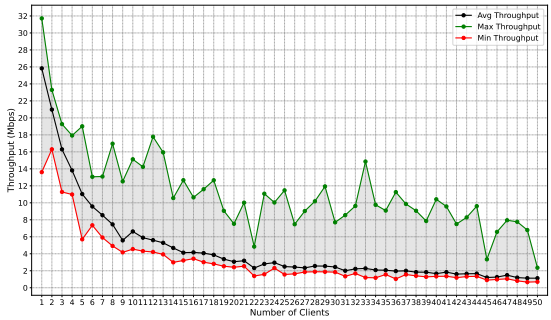
Latency results, however, present a more intricate scenario. While both protocols exhibit comparable average latencies in the same-connection configuration, the differences become more evident in the different-connection setup, whereas the number of clients



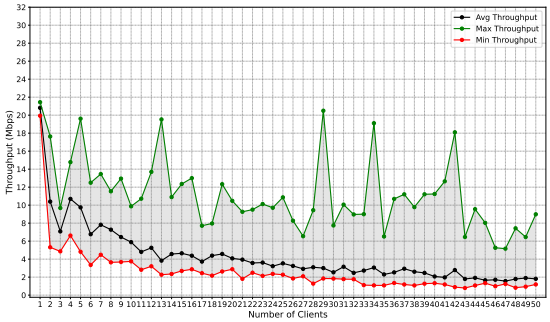
(a) TLS Same Connection



(b) TLS Different Connection

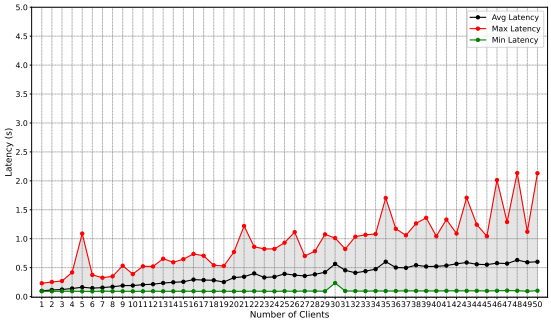


(c) QUIC Same Connection

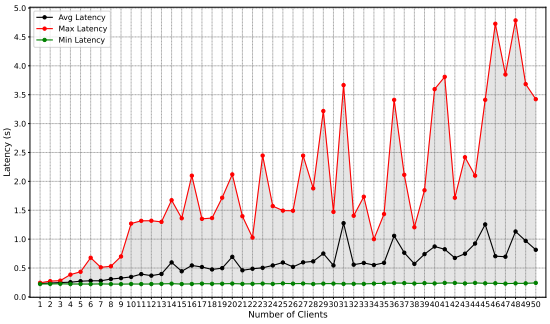


(d) QUIC Different Connection

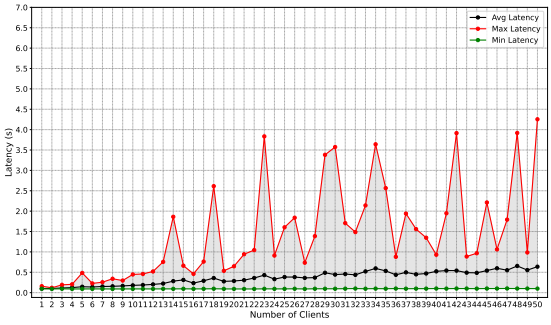
Figure 6.6: Throughput Variations with Different Number of Clients Across Diverse Protocols and Heuristics.



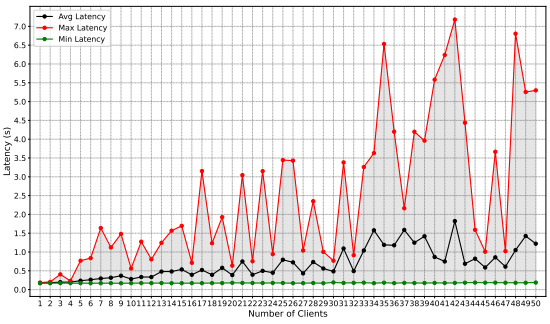
(a) TLS Same Connection



(b) TLS Different Connection



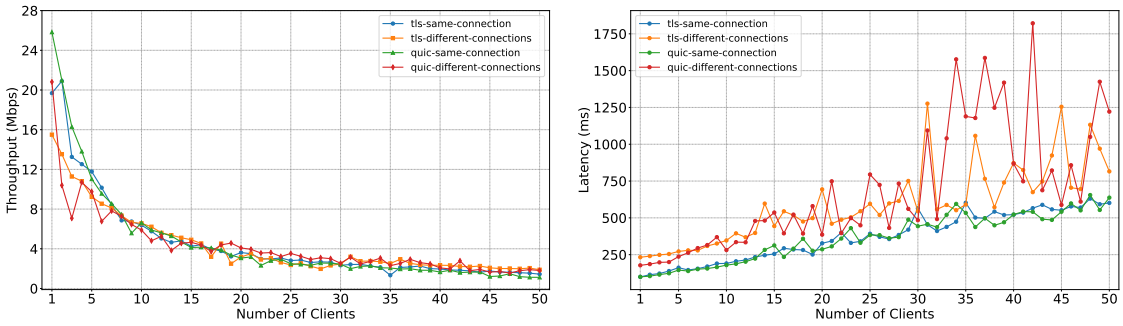
(c) QUIC Same Connection



(d) QUIC Different Connection

Figure 6.7: Latency Variations with Different Number of Clients Across Diverse Protocols and Heuristics.

increases, noticeable instability arises, with very pronounced fluctuations in latency, as depicted in Figure 6.8.



(a) Throughput (b) Latency

Figure 6.8: Average Throughput and Latency Variations with Different Number of Clients Across Diverse Protocols and Heuristics.

6.3 Resource

Resource allocation is a significant determinant of the efficiency and stability of network systems. This metric primarily involves the analysis of CPU or processing load and Memory allocation, which are essential for system operation under various conditions. By evaluating resource allocation, it is possible to determine how effectively the system manages its computational resources, particularly in scenarios characterized by heavy network traffic or constrained hardware capabilities. This analysis was performed across a client range from 1 to 50 to capture the system's resource utilization under different load conditions, and we measured the CPU and Memory usage every 0.1s using `psutil`[2], a python library, and each point in the plot represents the maximum resource value captured during each phase of testing.

Results will be presented for the gateway, relay node, and exit node. In the case of the gateway, running multiple independent clients on the same machine may not be a typical use case, but it showcases that a single computer could support that load, given the lightweight nature of the program. Nonetheless, the main focus of the analysis is on the relay node and exit node, where the impact of the number of clients needs to be studied. These nodes provide valuable insights into how client numbers affect system performance and resource allocation.

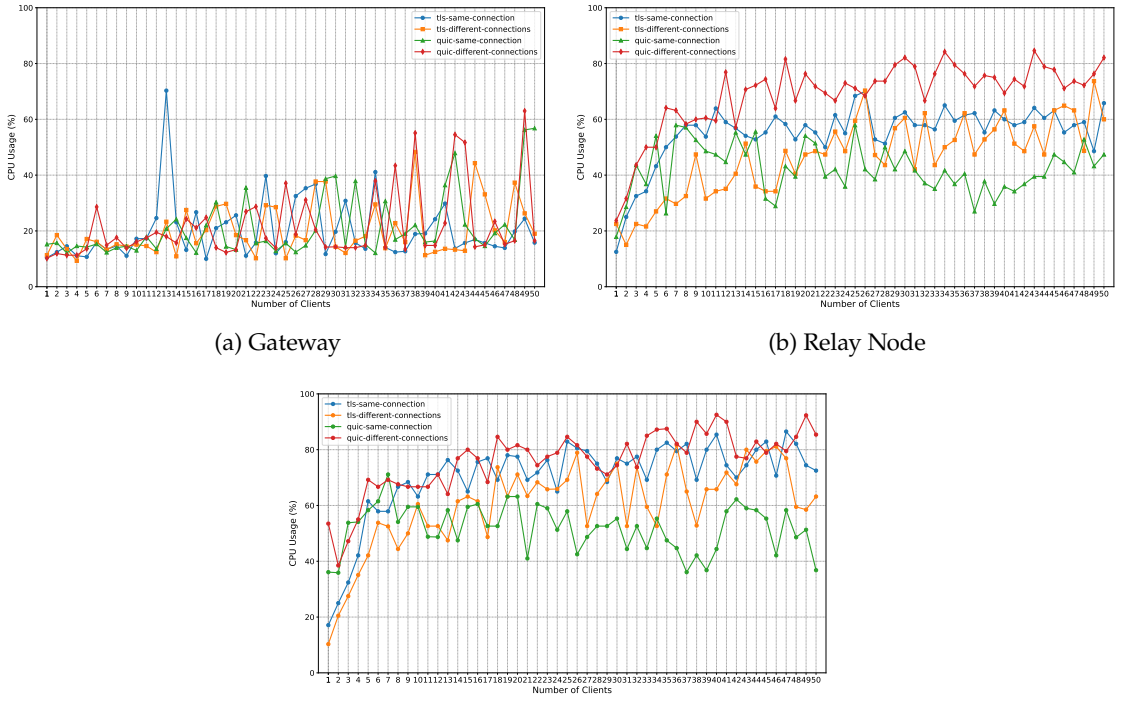
6.3.1 CPU or Processing Load

The CPU or processing load metric quantifies how much the system's central processing unit is utilized during operation. Elevated processing loads can indicate inefficiencies

[2]psutil manual - urlhttps://psutil.readthedocs.io/en/latest/# (Accessed on 16 September 2024)

within the system, potentially leading to increased latency and diminished overall performance. Therefore, continuous CPU usage monitoring and optimization were vital to ensure the system operates within acceptable performance parameters.

The results for CPU usage are depicted in Figure 6.9, highlighting how processing demands scale with client numbers. This analysis offers critical insights into the system's ability to manage varying client loads and maintain optimal performance.



(a) Gateway

(b) Relay Node

(c) Exit Node

Figure 6.9: Evaluating CPU Utilization Relative to the Number of Clients.

At the gateway, TLS exhibits moderate CPU usage for both the same-connection and different-connections scenarios, with some fluctuations as the number of clients increases. The CPU usage across different configurations remains reasonably similar, apart from a noticeable spike reaching 70%. This spike does not appear to follow any specific trend and is likely the result of an anomalous situation, possibly due to a background task. However, most values remain within the 10% to 50% range, reinforcing the idea that the software is lightweight. This suggests that even a regular personal computer can handle 50 clients while performing encryption computations and forwarding traffic without hitting any CPU bottlenecks.

A clear trend emerges as we shift to the relay node, with CPU usage rapidly increasing to the 60% to 80% range and then stabilizing. TLS shows a consistent rise in CPU usage as the number of clients grows. For the same-connection configuration, the CPU load starts at moderate levels but climbs significantly, reaching up to 65.8%. In the different-connections scenario, the increase in CPU usage is slower and more gradual. QUIC,

on the other hand, behaves differently: in the same-connection scenario, it maintains a lower CPU usage, achieving the best performance among all configurations. However, in the different-connections scenario, QUIC sees the highest CPU utilization, indicating that managing multiple QUIC connections at this stage significantly demands system resources, as expected.

At the exit node, CPU utilization becomes considerably high, especially in the QUIC different-connections and TLS same-connection configurations, where usage peaks at 92.5%. Once again, QUIC in the same-connection scenario proves to be the most efficient configuration in terms of performance. Notably, the slight increase in CPU usage at the exit node can primarily be attributed to handling traffic from file transfers initiated by servers, which involves final decryption, processing, and forwarding to the public internet. This stage is CPU-intensive, especially with many connections or large data flows, though the increase is not excessively high.

6.3.2 Memory

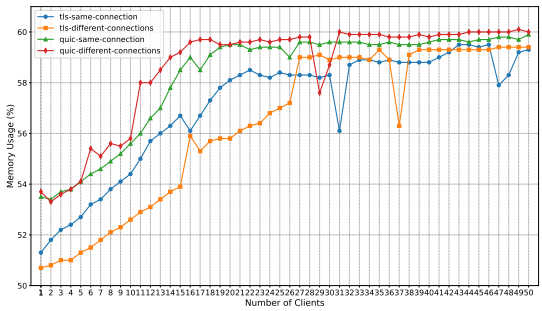
Memory allocation refers to the amount of system memory (RAM) required to sustain the system's operations. Effective memory management is crucial, especially in resource-limited environments, where excessive memory usage can result in system instability or degraded performance. This metric is instrumental in understanding how the system manages large datasets or high volumes of concurrent connections, thereby ensuring that the system remains stable and performs optimally under load.

The results for Memory usage are depicted in Figure 6.10, highlighting how memory demands scale with client numbers, providing critical insights into the system's ability to manage varying client loads and maintain optimal performance.

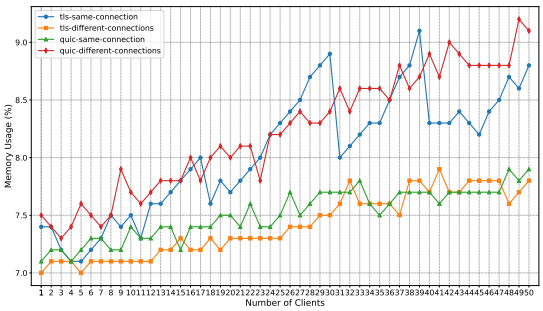
At the gateway, memory usage is relatively stable across all protocols and connection types, ranging from 50% to 60%, suggesting that the system can support large client loads without exhausting memory resources or experiencing performance bottlenecks. However, we can see that QUIC shows slightly higher memory usage than TLS when the client load is lower.

At the relay node (VPS), memory usage is way lower than at the gateway (personal computer), remaining steady across all protocols and connection types, typically ranging from 7% to 9%. Both TLS and QUIC exhibit similar memory consumption patterns, with QUIC showing a slightly higher memory demand, particularly in the different-connections scenario. Despite these minor differences, memory usage remains controlled, indicating that the relay node manages traffic efficiently, even as client numbers increase, without straining system resources.

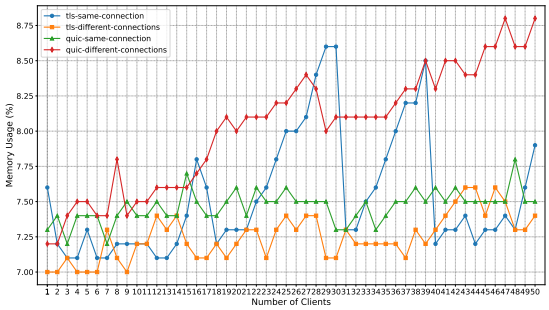
At the exit node, memory consumption shows slightly more variability but still falls within an acceptable range, between 7% and 9%. TLS and QUIC demonstrate stable memory usage, with QUIC different-connections configuration demanding slightly more memory, peaking at 8.8%. The exit node's role in decrypting and forwarding traffic to



(a) Gateway



(b) Relay Node



(c) Exit Node

Figure 6.10: Evaluating Memory Utilization Relative to the Number of Clients.

the public internet introduces some variability. Nevertheless, overall memory demands remain within manageable limits, ensuring that performance is not compromised even under higher client loads.

6.4 Resistance to Website Fingerprinting Attacks

In this section, we evaluate the resistance of our system, focusing on its ability to withstand website fingerprinting attacks, a form of traffic analysis where adversaries attempt to infer which websites a user visits by analyzing encrypted traffic patterns, as discussed in Section 2.2. The following subsections describe the testing environment for our experiments, the machine learning models used for analysis, and the evaluation results.

6.4.1 Testing Environment

To evaluate our system's resistance in the context of website fingerprinting, we designed a test environment based on established traffic analysis methodologies. We configured the network to use the QUIC protocol, multiplexing multiple connections within a single session (quic-same-connection). We applied a round-robin heuristic, switching circuits every 200 packets sent or received. The network followed a three-relay path, similar to the Tor architecture, with three distinct data-transmission circuits. Additionally, we did not

employ random jitter during data transmission, increasing the challenge to the system's resistance to traffic analysis, and we only used three paths, each with three relays, due to the number of available VPS to test our solution.

We selected 50 websites from the Alexa Top 1000 websites, a standard benchmark used in related work to evaluate systems against website fingerprinting attacks [49, 59, 124]. We sampled each website 20 times, generating 1,000 traffic samples (50 websites x 20 samples).

Traffic data was captured at the entry point of one circuit using tcpdump, which recorded all packet-level network activity during the browsing sessions, as depicted in Figure 6.11. This data included critical traffic features such as packet size, timing, and direction, which we later analyzed to evaluate resistance to fingerprinting attacks.

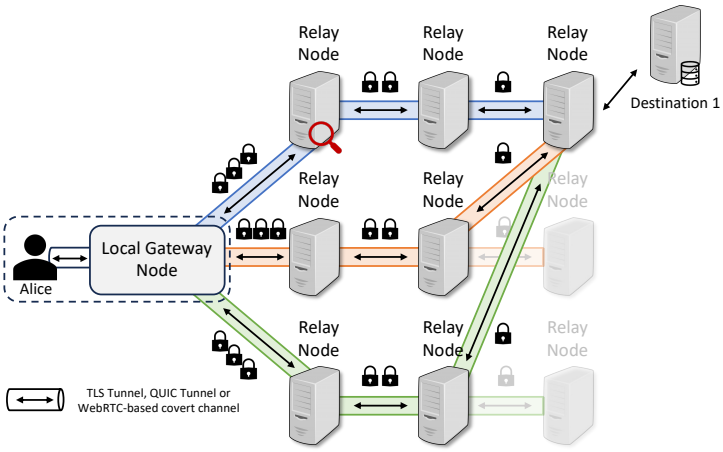


Figure 6.11: Test environment for the resistance to website fingerprinting attacks.

For automated browsing, we set up Selenium on Chrome Browser and disabled the browser's cache, ensuring each website loaded as if accessed for the first time. This configuration prevented caching effects from influencing the traffic and preserved the accuracy of the data we collected. We also disabled headless mode, allowing each webpage to render fully in the browser's graphical interface. This approach aims to mirror real-world browsing conditions, as headless browsing can omit certain rendering activities, leading to altered traffic patterns. After each webpage load, we introduced a 10-second delay to allow traffic to stabilize, ensuring that the traffic patterns we observed were primarily associated with the website load itself and not affected by background tasks.

6.4.2 Feature Extraction

To evaluate our system's resistance to website fingerprinting attacks, we extracted a comprehensive set of traffic features from the raw packet captures, focusing on those most relevant to distinguishing traffic patterns across different websites. The selection of these features was guided by established findings in prior traffic analysis studies and their potential to reveal distinctive browsing behaviours.

We began by extracting traffic volume features, including the total number of packets, total bytes, and the distribution of packet sizes (such as mean, standard deviation, and variance) for both inbound and outbound traffic. These metrics are essential for capturing each session's overall data flow characteristics. Complementing these volume features, we also focused on timing-related features, such as inter-arrival times between packets and burst sizes. The temporal structure of traffic, which can vary significantly between websites, is often reflected in these patterns. For instance, websites may load content in bursts, and capturing this behaviour helps differentiate between websites.

We also included meta-features such as time-to-live (TTL) values and TCP window sizes to supplement the traffic and protocol-specific data. These features offer insights into the characteristics of the network paths and connections involved, adding another layer of information to the traffic analysis.

All features were extracted using packet-level captures with tools like tcpdump and pyshark and processed using Python. Where necessary, features were normalized to account for differences in scale, ensuring that no single feature would disproportionately influence the machine learning models used for classification. This comprehensive approach to feature extraction ensured a balanced and accurate representation of traffic characteristics, enhancing the robustness of our evaluation.

6.4.3 Machine-Learning Models Used

We utilized a diverse set of widely used machine learning models in traffic analysis research to evaluate the system's resistance to website fingerprinting. These models include Naïve Bayes, Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Random Forest, and Extra Trees. We employed these models with their default configurations without performing hyperparameter tuning to ensure consistency and provide a fair benchmark. We trained the models on the collected dataset, using 80% of the data for training and reserving the remaining 20% for testing, replicating a close-world evaluation. In a close-world evaluation, the classifier is trained and tested on a fixed, predefined set of websites. In contrast, in an open world, the classifier is trained on a subset of websites and then tested with both "known" (seen in training) and "unknown" (new, unseen) websites. We also used advanced models such as Gradient Boosting and XGBoost to further explore complex traffic analysis patterns. Below is a brief description of the models we used:

Naïve Bayes: A probabilistic classifier that applies Bayes' theorem and assumes a strong assumption of feature independence (naive). Its interpretability makes it a useful first step in assessing basic traffic features.

Logistic Regression: A linear classification model that predicts the probability of a binary outcome based on a logistic function. It is frequently used for its simplicity and

interpretability in classification tasks, especially when the data is linearly separable.

K-Nearest Neighbors (KNN): A non-parametric, instance-based learning algorithm that predicts by finding the most common class among the k-nearest neighbours of a data point. It is particularly valuable when the structure of the data is unknown or highly non-linear.

Support Vector Machines (SVM): A robust classification algorithm that finds the optimal hyperplane to best separate data points of different classes in high-dimensional space. It excels in scenarios where data is not linearly separable by transforming input features via kernel functions.

Random Forest: An ensemble learning method that builds multiple decision trees during training and outputs the class that represents the majority vote from individual trees. Random Forest reduces overfitting and improves generalization by introducing randomness in both feature selection and sample selection, and it is particularly useful for finding intricate features by capturing different patterns and variations across different labels.

Extra Trees: Similar to Random Forest, but it introduces further randomness by selecting random thresholds for splitting trees, which helps reduce variance and improve predictive performance, especially in noisy datasets.

Gradient Boosting: An ensemble learning method that sequentially builds models, with each new model correcting errors made by the previous ones. It is highly effective in capturing complex patterns through iterative improvements. This model helps detect nuanced traffic characteristics that may evade simpler classifiers, such as subtle variations in packet size or timings.

XGBoost: An optimized version of Gradient Boosting, known for its computational efficiency and superior performance. It incorporates regularization techniques that prevent overfitting and improve model generalization, making it particularly suitable for large datasets. Its ability to model complex, non-linear patterns efficiently enables it to uncover subtle patterns.

6.4.4 Extracted Metrics

We measured the effectiveness of each model using several key metrics: accuracy, precision, recall, F1-score, Matthews Correlation Coefficient, Cohen's Kappa Score, Log Loss, Jaccard Score and F2-score. These metrics allowed us to evaluate the system's ability to obscure traffic patterns and resist fingerprinting attacks. Each metric offers a distinct perspective on model performance, ensuring a comprehensive evaluation, particularly for multi-class classification.

Accuracy: The proportion of correct predictions (True Positives and True Negatives) made by the model out of all predictions. It is a straightforward and commonly used metric to assess a model's overall performance if the dataset is balanced. Regarding Website Fingerprinting, a multi-class setting, accuracy provides a clear picture of how well the model can distinguish between different websites based on traffic patterns.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP refers to true positives (correctly predicted websites), TN refers to true negatives (correctly identified non-website classes), FP refers to false positives (incorrectly identified websites), and FN refers to false negatives (missed correct websites). Since our dataset is balanced (each class has an equal number of samples), accuracy offers a good initial overview of the model's performance. However, it does not account for how well the model performs for each class, where precision, recall, and F1-score come into play.

Weighted Precision: Precision measures the proportion of true positives among all predicted positives, providing insight into how well the model avoids false positives. In a multi-class setting, weighted precision calculates the average precision across all classes, considering the class imbalance in the dataset. In our case, as each website has an equal number of samples and is balanced, the precision evaluates the model's accuracy in correctly identifying websites without disproportionately favoring one class over another.

$$\text{Precision}_i = \frac{TP_i}{TP_i + FP_i}$$

Weighted Recall: Recall (also known as sensitivity) measures the proportion of true positives out of all actual instances of a class, reflecting the model's ability to avoid false negatives (missed identifications). Like precision, recall is straightforward to interpret in this balanced dataset because each class contributes equally, and there's no need to adjust for different class sizes.

$$\text{Recall}_i = \frac{TP_i}{TP_i + FN_i}$$

Weighted F1-Score: The harmonic mean of precision and recall, this metric balances the two metrics to assess the model's overall performance, particularly in handling imbalanced data. Since the dataset is balanced, each class has equal representation, and the F1-score can be computed directly without any weighting adjustments.

$$\text{F1-Score}_i = 2 \cdot \frac{\text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

Matthews Correlation Coefficient (MCC): A robust metric that considers all four confusion matrix components (TP, TN, FP, FN) to provide a balanced view of model performance.

It is particularly valuable in multi-class classification, as it handles both positive and negative predictions across all classes. It provides a score between -1 and 1, where 1 indicates perfect prediction, 0 indicates random performance, and -1 signifies complete disagreement.

$$\text{MCC} = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}$$

Cohen's Kappa: Cohen's Kappa measures the agreement between the model's predictions and the actual class labels, taking into account the possibility of agreement occurring by chance. Kappa is useful for ensuring that the model's performance is not simply the result of random guessing. It ranges from -1 to 1, where 1 represents perfect agreement, 0 agreement equal to random chance, and -1 complete disagreement.

$$\kappa = \frac{P_o - P_e}{1 - P_e}$$

Here, P_o is the observed agreement, and P_e is the expected agreement, calculated as the probability of random agreement based on the class distribution.

Log Loss: It measures the uncertainty of the model's predictions by measuring the likelihood of the model's predicted probabilities compared to the actual class labels. It penalizes incorrect predictions based on the confidence of the model, providing a more nuanced view of the model's performance than accuracy alone. Lower log loss values indicate more confident and accurate predictions.

$$\text{Log Loss} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(p_{i,c})$$

Where, N is the number of samples, C is the number of classes, $y_{i,c}$ is a binary indicator (1 if class c is the correct label for instance i, and 0 otherwise), and $p_{i,c}$ is the predicted probability of instance i belonging to class c.

Weighted Jaccard Score: The Jaccard Score is a metric used to assess the similarity between the predicted and true classes. It is calculated as the size of the intersection divided by the size of the union of the predicted and true classes. For multi-class classification, the Weighted Jaccard Score calculates the score for each class and weighs it by the number of true instances in each class.

$$\text{Jaccard}_i = \frac{TP_i}{TP_i + FP_i + FN_i}$$

Weighted F2-Score: The F2-Score is a variant of the F1-Score that places more emphasis on recall than precision. It is particularly useful when recall is more critical than precision, as in scenarios where false negatives are more detrimental than false positives.

$$\text{F2-Score}_i = \frac{(1+2^2)\cdot \text{Precision}_i \cdot \text{Recall}_i}{2^2 \cdot \text{Precision}_i + \text{Recall}_i}$$

6.4.5 Results and Discussion

The results from our experiments show that the system exhibits a strong level of resistance under most conditions, as demonstrated by the performance of the initial machine learning models (Naïve Bayes, Logistic Regression, K-Nearest Neighbors, Support Vector Machines, Random Forest, and Extra Trees). The accuracy - considered to be the main metric in website fingerprinting attacks [105] - of these models in identifying specific websites in the close world was relatively low, indicating that the system effectively obscures traffic patterns and resists website fingerprinting attacks, as depicted in Figure 6.12, even under less-than-ideal conditions. Moreover, Table 6.2 presents the other key metrics abovementioned. However, more advanced models like Gradient Boosting and XGBoost achieved higher classification accuracy, shown in Figure 6.13 and complementary in Table 6.3, suggesting that the system's protection decreases under more sophisticated traffic analysis techniques.

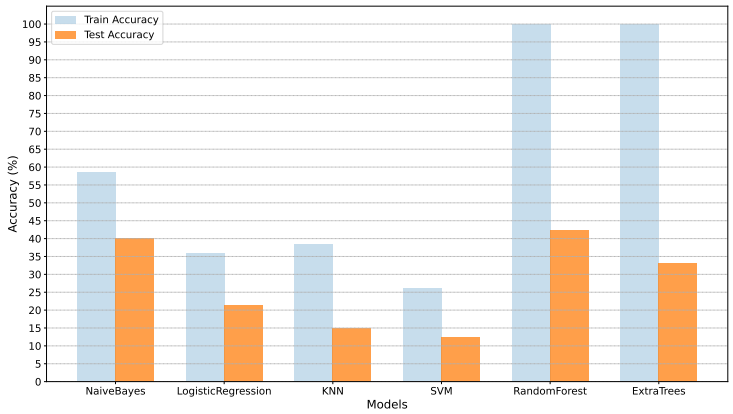


Figure 6.12: Train and Test Accuracy on Various Machine Learning Models.

Model	Weighted Precision	Weighted Recall	Weighted F1-Score	Matthews Correlation Coefficient	Cohen's Kappa Score	Log Loss	Weighted Jaccard Score	Weighted F2-Score
NaïveBayes	0.4688	0.4000	0.3944	0.3939	0.3878	18.0874	0.3006	0.3881
LogisticRegression	0.2298	0.2150	0.1945	0.2006	0.1990	2.9339	0.1245	0.2007
KNN	0.1312	0.1500	0.1315	0.1334	0.1327	18.8746	0.0789	0.1394
SVM	0.1209	0.1250	0.1045	0.1093	0.1071	—	0.0676	0.1116
RandomForest	0.4206	0.4250	0.4009	0.4141	0.4133	3.5340	0.2815	0.4112
ExtraTrees	0.3283	0.3300	0.3100	0.3171	0.3163	4.4050	0.2069	0.3176

Table 6.2: Complementary Metrics for Various Machine Learning Models.

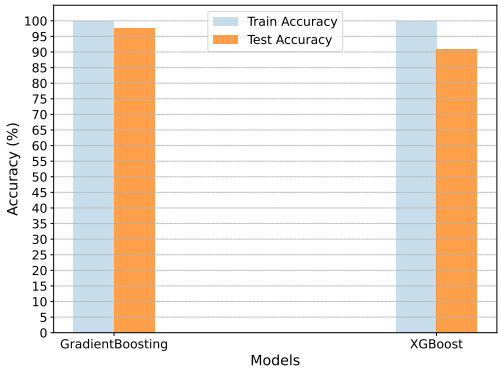


Figure 6.13: Train and Test Accuracy on More Complex Machine Learning Models.

Model	Weighted Precision	Weighted Recall	Weighted F1-Score	Matthews Correlation Coefficient	Cohen's Kappa Score	Log Loss	Weighted Jaccard Score	Weighted F2-Score
GradientBoosting	0.9711	0.9600	0.9599	0.9593	0.9590	0.2312	0.9640	0.9793
XGBoost	0.9063	0.8700	0.8699	0.8680	0.8670	0.3580	0.8542	0.9075

Table 6.3: Complementary Metrics for More Complex Machine Learning Models.

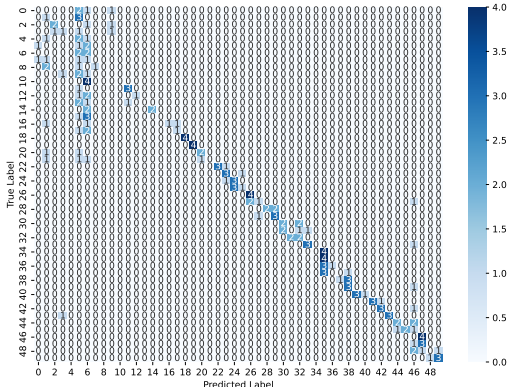
Misclassification Analysis

We also present confusion matrices using a seaborn heatmap for each model in Figure 6.14 and Figure 6.15 to better understand the misclassification patterns. The confusion matrix helps illustrate how well the system conceals traffic by showing which websites were incorrectly classified. This highlights any websites that are particularly vulnerable to fingerprinting and provides a deeper evaluation of the system's performance beyond standard accuracy metrics.

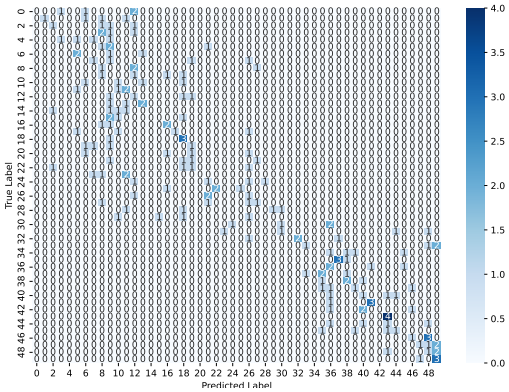
Across the models, Logistic Regression, KNN and SVM exhibit the highest misclassification rates, reflecting their challenges in accurately distinguishing traffic patterns. Random Forest, Extra Trees, and Naïve offer intermediate performance, with fewer misclassifications but still some vulnerabilities. In contrast, Gradient Boosting and XGBoost show superior performance with minimal misclassification, indicating higher accuracy in traffic identification.

Feature Importance

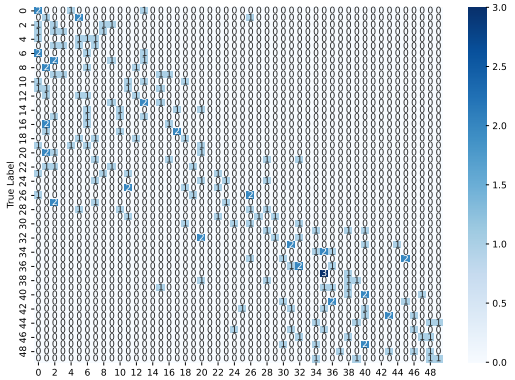
When analyzing the feature importance of the models that produced the best results, it became clear that certain features consistently played a pivotal role in driving predictions. Notably, session duration (i.e., the duration between the first and the last packet in the capture) emerged as the most influential feature in Gradient Boosting and XGBoost, highlighting its critical impact on network behaviour classification. However, while Gradient Boosting primarily relied on session duration, XGBoost demonstrated a more balanced use of features, incorporating additional metrics such as total packets, burst size variability, and window size statistics. This broader feature utilization in XGBoost



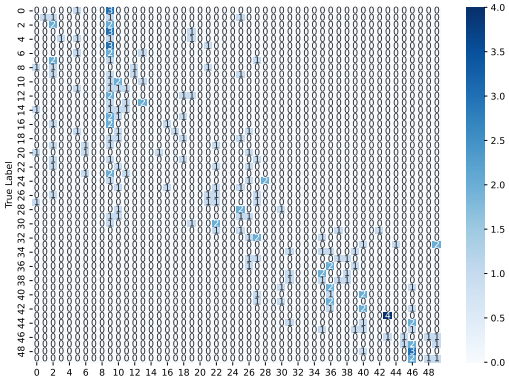
(a) NaïveBayes



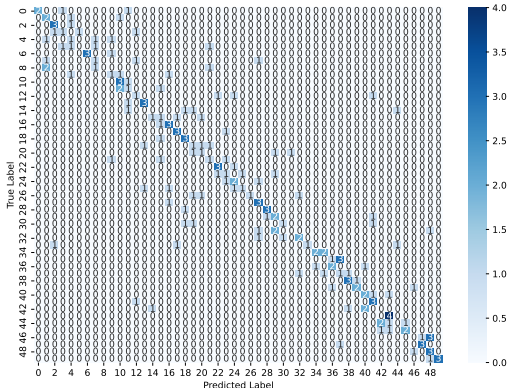
(b) LogisticRegression



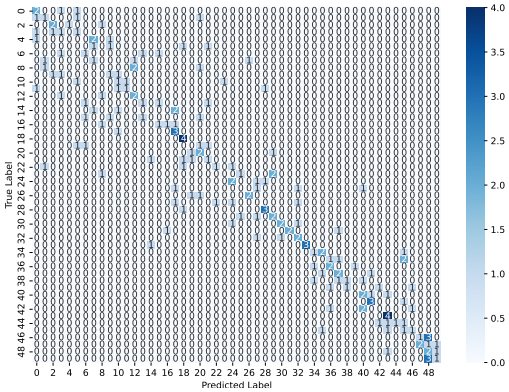
(c) KNN



(d) SVM

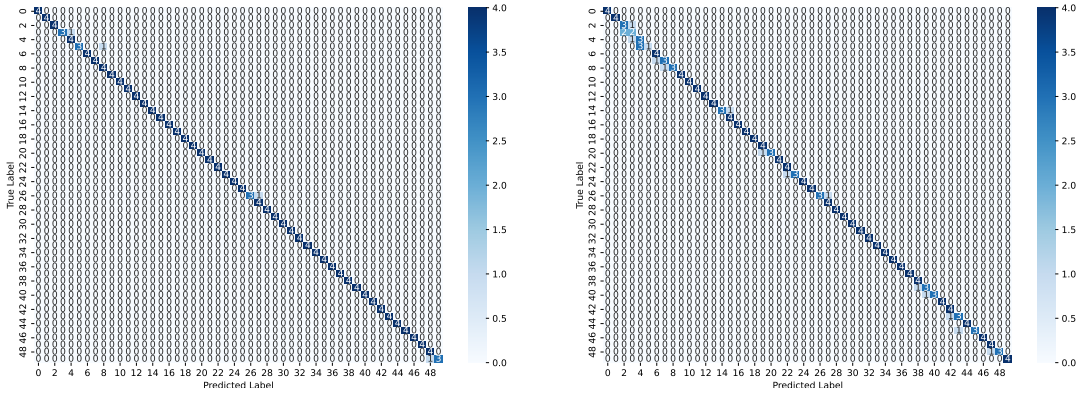


(e) RandomForest



(f) ExtraTrees

Figure 6.14: Heatmap of Confusion Matrix for Various Machine Learning Models.



(a) GradientBoosting (b) XGBoost

Figure 6.15: Heatmap of Confusion Matrix for More Complex Machine Learning Models.

reflects its ability to capture more nuanced patterns within the data despite slightly lower accuracy compared to Gradient Boosting.

Evaluation Limitations & Strategies for Enhanced Protection

The controlled nature of our testing environment played a key role in these findings, and the accuracy of such evaluations is known to be sensitive to the underlying conditions of the network segments under analysis [105]. By accessing websites repeatedly on the same day under consistent network conditions, we ensured stable traffic patterns, which most likely have significantly influenced feature importance and model behaviour. However, this also limited the variability typically observed in real-world networks, where fluctuating conditions could introduce noise and potentially aid in concealing traffic. Moreover, the lack of random jitter, the small number of distinct paths (resulting from the number of VPS available to test the system), the heuristic applied (round-robin heuristic instead of random selection, proven to yield worse results for the classifiers [120]) and the use of a relatively high threshold of 200 packets (instead of fewer packets like 50 or 100, also proven to yield worse results for the classifiers [120]) before switching circuits, made the system more vulnerable to traffic analysis attacks.

Taking the insight into account, we hypothesize that if the network conditions were made more variable - by capturing traffic traces under different conditions, activating random jitter, and increasing the frequency of path changes by lowering the packet threshold - the models' performance would degrade. Increased fluctuations would obscure key patterns, introduce additional noise, and complicate feature interpretation, ultimately making accurate predictions more challenging. Since session duration was identified as the most important feature in the best models, we believe that the introduction of such variability in the network conditions would reduce its relevance, forcing the models to rely more heavily on other features, thus making the system less observable.

Exploring the WebRTC Potential of MIRACE

Given the abovementioned challenges, one promising approach is using WebRTC encapsulation, as implemented in the MIRACE system. This method encapsulates traffic within the video frames of WebRTC communication streams, effectively camouflaging it and making it indistinguishable from regular WebRTC traffic. By doing so, it significantly hinders detection efforts by censors. It has been shown that this encapsulation method is resilient against both active and passive correlation attacks [11, 114], making it a promising option for maintaining unobservability in environments where detection and censorship are a significant concern.

Despite the lack of handling packet loss and retransmission in the current implementation, the current configuration can still function effectively in scenarios where packet loss and retransmission are not critical, for instance, in real-time communication applications or in client applications that support packet loss. However, we are awaiting a new version of TorKameleon to be released, relying on Turbo Tunnel [39] for the WebRTC encapsulation method, to then integrate it on MIRACE to make the system more reliable across a broader range of scenarios, including data integrity and stability.

6.5 Summary

This chapter presented a comprehensive experimental evaluation of MIRACE, focusing on its performance, scalability, and resource efficiency under various conditions. Our analysis revealed significant insights into the system's behaviour across different metrics. Regarding performance, we examined the impact of the number of nodes per circuit on throughput and latency and the bootstrap time during the key exchange protocol. The findings indicate that it maintains satisfactory throughput and latency, even as the number of nodes increases.

Scalability tests further demonstrated the system's ability to handle various clients simultaneously efficiently, maintaining throughput in the orders of Mbps even under 50 clients concurrently using the same path. On the other hand, fluctuations in latency became more common when this increase happened.

Regarding resource usage, we analyzed the system's CPU and memory consumption, discovering that it introduces a moderate processing overhead, especially during peak traffic periods. Memory utilization remained consistent across various test scenarios, though slight fluctuations were noted when managing more intensive operations.

Our evaluation also included assessing the system's robustness against website fingerprinting attacks. We used eight machine learning models and extracted several metrics commonly used to evaluate the model's performance. The results indicate that while the system resists many machine learning models, GradientBoosting and XGBoost achieve higher accuracy in traffic identification. However, we are confident that MIRACE can increase its resistance through the insights shared in Section 6.4.5.

In conclusion, the experimental evaluation confirmed that the proposed solution successfully meets the performance, scalability, and resource efficiency objectives. Furthermore, it draws insights into how it can resist traffic analysis attacks.

7

Conclusions

This dissertation comprehensively delved into the critical challenges surrounding anonymity and privacy within anonymization networks, with a particular focus on the Tor network's vulnerabilities. As mentioned previously, totalitarian regimes employ advanced surveillance and censorship techniques, utilizing sophisticated traffic analysis and machine learning tools to compromise the privacy offered by current solutions and highlighting the need to improve the existing system further or develop new ones that are more resilient and can effectively protect users' anonymity.

In response to these growing concerns, we proposed and implemented MIRACE, a censorship-resistant anonymization network to enhance user privacy and anonymity. By utilizing dynamic multipath routing and a diverse range of tunnelling protocols, we present a novel solution that strengthens resistance to modern threats. The system integrates covert channels, multi-layered encryption through Onion Routing, and protocols such as TLS, QUIC, and WebRTC to ensure user communications' privacy and untraceability. MIRACE uses these layered strategies to make communications significantly more resilient against traffic analysis, fingerprinting, and correlation attacks.

One of the central innovations of our system is its dynamic multipath routing approach. Unlike static routing strategies, our tool dynamically selects and reroutes traffic across multiple paths at the packet level. This dynamicity, combined with protocol diversity, complicates the task of adversaries seeking to correlate and trace communications back to individual users. Moreover, integrating techniques like random jitter and traffic encapsulation ensures that MIRACE offers stronger resistance to censorship and surveillance.

Through extensive experimental validation, we have confirmed that MIRACE effectively balances privacy and performance. Even under conditions with multiple nodes and clients, the system maintained acceptable throughput and latency, making it suitable for diverse application contexts. Importantly, our system demonstrated robustness against various traffic analysis models, and following according to Vilalonga et al. [114], has a WebRTC encapsulation method that is resilient against both active and passive correlation attacks, proving its potential as a privacy-preserved anonymization tool. Our findings indicate that MIRACE presents a viable alternative to existing solutions, offering enhanced

privacy protections while meeting reasonable performance.

7.1 Main Contributions

Our main contribution is the development of the MIRACE system. The prototype can be used by the scientific community and practitioners to investigate the developed mechanisms further and extend the baseline solution or to help develop other systems to protect against Internet censorship. Our additional contributions include (1) a comprehensive analysis of MIRACE's performance under various configurations and parameters, assessing both throughput and latency in real-world conditions; (2) a detailed study of MIRACE's resistance against website fingerprinting attacks, particularly modern traffic analysis models utilizing machine learning techniques; (3) a system that achieves competitive throughput and latency performance when compared to existing anonymization networks, in the order of Mbps; (4) an examination of the computational resource impact of MIRACE, ensuring that its privacy-preserving mechanisms do not significantly compromise system efficiency; and (5) an enhanced solution that strengthens the resilience of anonymization networks against deanonymization attacks, mainly through its use of dynamic multipath routing and protocol diversity, which mitigate traffic correlation and machine learning-based attacks.

7.2 Future Work

The work developed in this thesis allows us to identify various potential directions for future work, including but not limited to the following:

- Incorporating the New WebRTC Encapsulation Version with TurboTunnel: Future work should explore integrating the latest version of TorKameleon's WebRTC encapsulation with TurboTunnel. TurboTunnel addresses packet loss and retransmission issues commonly encountered with WebRTC, particularly in regular scenarios of poor network conditions.

- Proxy Distribution: The system should also explore ways to distribute proxy information to users in a censorship-resistant manner, following methods such as those discussed in recent research [14, 115], addressing an essential part of the broader challenge known as the Bridge Distribution Problem.

- Comprehensive Evaluation with Advanced Configurations: Further evaluation with different configurations could yield insights into performance and privacy improvements. This includes evaluating the system with more frequent path and packet changes and varying levels of random jitter. Moreover, testing these configurations against other types of attacks, active and passive, could provide a more comprehensive understanding of the system's robustness.

- Incorporating Differential Privacy Mechanisms: Implementing a differential privacy mechanism could add another layer of anonymity to the system by obfuscating the statistical properties of the user's network activity. By carefully introducing controlled noise into the traffic data, differential privacy could ensure that individual user behaviours are indistinguishable from the aggregate data set, thus significantly enhancing resistance to statistical analysis and behavioural fingerprinting.

Bibliography

[1] P. Agrawal et al. "Traceable mixnets". In: *arXiv preprint arXiv:2305.08138* (2023) (cit. on p. 14).

[2] 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 (cit. on p. 11).

[3] M. AlSabah et al. "The path less travelled: Overcoming Tor's bottlenecks with traffic splitting". In: *Privacy Enhancing Technologies: 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings 13*. Springer. 2013, pp. 143–163 (cit. on pp. 12, 29).

[4] K. Amit and S. Neeraj. "Privacy preservation in big data using k-anonymity algorithm with privacy key". In: *International Journal Of Computer Applications* 153.5 (2016), pp. 0975–8887 (cit. on p. 18).

[5] R. Attarian et al. "MixFlow: Assessing Mixnets Anonymity with Contrastive Architectures and Semantic Network Information". In: *Cryptology ePrint Archive* (2023) (cit. on pp. 9, 28).

[6] L. Barman et al. "PriFi: Low-latency anonymity for organizational networks". In: *arXiv preprint arXiv:1710.10237* (2017) (cit. on pp. 14, 15).

[7] L. Barman et al. "Groove: Flexible {Metadata-Private} Messaging". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 2022, pp. 735–750 (cit. on pp. 14, 17).

[8] D. Barradas and N. Santos. "Towards a Scalable Censorship-Resistant Overlay Network Based on WebRTC Covert Channels". In: *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good*. DICG'20. Delft, Netherlands: Association for Computing Machinery, 2021, pp. 37–42. ISBN: 9781450381970. DOI: 10.1145/3428662.3428788. URL: https://doi.org/10.1145/3428662.3428788 (cit. on p. 25).

[9] D. Barradas, N. Santos, and L. Rodrigues. "Effective detection of multimedia protocol tunneling using machine learning". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 169–185 (cit. on pp. 23–25, 28).

[10] D. Barradas, N. Santos, and L. E. Rodrigues. "DeltaShaper: Enabling Unobservable Censorship-resistant TCP Tunneling over Videoconferencing Streams." In: *Proc. Priv. Enhancing Technol.* 2017.4 (2017), pp. 5–22 (cit. on pp. 23, 28).

[11] D. Barradas et al. "Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 35–48. ISBN: 9781450370899. DOI: 10.1145/3372297.3417874. URL: https://doi.org/10.1145/3372297.3417874 (cit. on pp. 2, 25, 26, 28, 101).

[12] D. M. B. Barradas. "Unobservable covert streaming for internet censorship circumvention". Master's thesis, Universidade de Lisboa, 2016 (cit. on p. 23).

[13] I. Ben Guirat, D. Gosain, and C. Diaz. "Mixim: Mixnet design decisions and empirical evaluation". In: *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*. 2021, pp. 33–37 (cit. on p. 14).

[14] C. Bocovich et al. "Snowflake, a censorship circumvention system using temporary WebRTC proxies". In: *USENIX Security Symposium*. USENIX, 2024. URL: https://www.usenix.org/system/files/sec24fall-prepub-1998-bocovich.pdf (cit. on pp. 26, 28, 104).

[15] A. Campan and T. M. Truta. "Data and structural k-anonymity in social networks". In: *International Workshop on Privacy, Security, and Trust in KDD*. Springer. 2008, pp. 33–54 (cit. on p. 18).

[16] D. L. Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". In: *Communications of the ACM* 24.2 (1981), pp. 84–90 (cit. on pp. 1, 13, 17).

[17] C. Chen et al. "HORNET: High-speed onion routing at the network layer". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 1441–1454 (cit. on pp. 1, 6).

[18] C. Chen et al. "TARANET: Traffic-analysis resistant anonymity at the network layer". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 137–152 (cit. on pp. 14, 15).

[19] G. Cherubin, R. Jansen, and C. Troncoso. "Online Website Fingerprinting: Evaluating Website Fingerprinting Attacks on Tor in the Real World". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, 2022, pp. 753–770. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/cherubin (cit. on p. 2).

[20] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. "Riposte: An anonymous messaging system handling millions of users". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 321–338 (cit. on p. 1).

[21] W. Cui, T. Chen, and E. Chan-Tin. "More Realistic Website Fingerprinting Using Deep Learning". In: *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. 2020, pp. 333–343. DOI: 10.1109/ICDCS47774.2020.00058 (cit. on p. 10).

[22] G. Danezis, R. Dingledine, and N. Mathewson. "Mixminion: design of a type III anonymous remailer protocol". In: *2003 Symposium on Security and Privacy, 2003.* 2003, pp. 2–15. DOI: 10.1109/SECPRI.2003.1199323 (cit. on pp. 5, 6).

[23] G. Danezis and I. Goldberg. "Sphinx: A compact and provably secure mix format". In: *2009 30th IEEE Symposium on Security and Privacy*. IEEE. 2009, pp. 269–282 (cit. on p. 13).

[24] A. Darer, O. Farnan, and J. Wright. "FilteredWeb: A framework for the automated search-based discovery of blocked URLs". In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. 2017, pp. 1–9. DOI: 10.23919/TMA.2017.8002914 (cit. on p. 1).

[25] D. Das et al. "Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two". In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 108–126 (cit. on pp. 2, 5, 31).

[26] D. Das et al. "Divide and funnel: a scaling technique for mix-networks". In: *Cryptology ePrint Archive* (2021) (cit. on p. 12).

[27] W. De la Cadena et al. "Analysis of multi-path onion routing-based anonymization networks". In: *Data and Applications Security and Privacy XXXIII: 33rd Annual IFIP WG 11.3 Conference, DBSec 2019, Charleston, SC, USA, July 15–17, 2019, Proceedings 33*. Springer. 2019, pp. 240–258 (cit. on p. 12).

[28] W. De la Cadena et al. "TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1971–1985. ISBN: 9781450370899. DOI: 10.1145/3372297.3423351. URL: https://doi.org/10.1145/3372297.3423351 (cit. on pp. 11, 12, 27, 29).

[29] R. Deshmukh et al. "Video Conferencing using WebRTC". In: *2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*. IEEE. 2023, pp. 857–864 (cit. on p. 24).

[30] C. Diaz, H. Halpin, and A. Kiayias. "The Nym Network". In: (2021) (cit. on pp. 8, 14–17).

[31] T. Dierks and E. Rescorla. *The transport layer security (TLS) protocol version 1.2*. Tech. rep. 2008 (cit. on p. 28).

[32] R. Dingledine, N. Mathewson, P. F. Syverson, et al. "Tor: The second-generation onion router." In: *USENIX security symposium*. Vol. 4. 2004, pp. 303–320 (cit. on pp. 1, 6, 7, 28).

[33] J. R. Douceur. "The sybil attack". In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260 (cit. on p. 10).

[34] K. P. Dyer, S. E. Coull, and T. Shrimpton. "Marionette: A Programmable Network Traffic Obfuscation System". In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, 2015-08, pp. 367–382. ISBN: 978-1-939133-11-3. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/dyer (cit. on p. 2).

[35] K. Elmenhorst et al. "Web censorship measurements of HTTP/3 over QUIC". In: *Proceedings of the 21st ACM Internet Measurement Conference*. 2021, pp. 276–282 (cit. on p. 22).

[36] R. Ensafi et al. "Examining How the Great Firewall Discovers Hidden Circumvention Servers". In: *Proceedings of the 2015 Internet Measurement Conference*. IMC '15. Tokyo, Japan: Association for Computing Machinery, 2015, pp. 445–458. ISBN: 9781450338486. DOI: 10.1145/2815675.2815690. URL: https://doi.org/10.1145/2815675.2815690 (cit. on p. 1).

[37] Y. Feng et al. "A study of china's censorship and its evasion through the lens of online gaming". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 2599–2616 (cit. on pp. 1, 28).

[38] D. Fifield. "Threat modeling and circumvention of Internet censorship". PhD thesis. EECS Department, University of California, Berkeley, 2017. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-225.html (cit. on p. 23).

[39] D. Fifield. "Turbo Tunnel, a good way to design censorship circumvention protocols". In: *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*. USENIX Association, 2020-08. URL: https://www.usenix.org/conference/foci20/presentation/fifield (cit. on pp. 84, 101).

[40] G. Figueira, D. Barradas, and N. Santos. "Stegozoa: Enhancing WebRTC Covert Channels with Video Steganography for Internet Censorship Circumvention". In: 2022-05. DOI: 10.1145/3488932.3517419 (cit. on pp. 2, 25, 26, 28, 51).

[41] T. Fletcher and A. Hayes-Birchler. "Comparing Measures of Internet Censorship: Analyzing the Tradeoffs between Expert Analysis and Remote Measurement". In: 2020-07. DOI: 10.5281/zenodo.3967397 (cit. on pp. 1, 28).

[42] G. Forecast et al. "Cisco visual networking index: global mobile data traffic forecast update, 2017–2022". In: *Update* 2017 (2019), p. 2022 (cit. on p. 23).

[43] A. Freier, P. Karlton, and P. Kocher. *The secure sockets layer (SSL) protocol version 3.0*. Tech. rep. 2011 (cit. on p. 28).

[44] J. Geddes, M. Schuchard, and N. Hopper. "Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. CCS '13. Berlin, Germany: Association for Computing Machinery, 2013, pp. 361–372. ISBN: 9781450324779. DOI: 10.1145/2508859.2516742. URL: https://doi.org/10.1145/2508859.2516742 (cit. on pp. 23, 24, 28).

[45] Z. Guan et al. "An empirical analysis of plugin-based tor traffic over SSH tunnel". In: *MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM)*. IEEE. 2019, pp. 616–621 (cit. on p. 28).

[46] Z. Guan et al. "FlowTracker: Improved flow correlation attacks with denoising and contrastive learning". In: *Computers & Security* 125 (2023), p. 103018 (cit. on pp. 2, 9, 28).

[47] I. B. Guirat and C. Diaz. "Mixnet optimization methods". In: *Proceedings on Privacy Enhancing Technologies* (2022) (cit. on pp. 14, 15).

[48] 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 (cit. on p. 5).

[49] J. Hayes and G. Danezis. "k-fingerprinting: A robust scalable website fingerprinting technique". In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016, pp. 1187–1203 (cit. on pp. 10, 92).

[50] S. Henri et al. "Protecting against website fingerprinting with multihoming". In: *Proceedings on Privacy Enhancing Technologies* (2020) (cit. on pp. 2, 11, 27).

[51] A. Houmansadr, C. Brubaker, and V. Shmatikov. "The Parrot Is Dead: Observing Unobservable Network Communications". In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 65–79. DOI: 10.1109/SP.2013.14 (cit. on pp. 2, 24).

[52] A. Houmansadr et al. "IP over Voice-over-IP for censorship circumvention". In: *arXiv preprint arXiv:1207.2683* (2012) (cit. on pp. 23, 28).

[53] M. Husák et al. "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting". In: *EURASIP Journal on Information Security* 2016 (2016), pp. 1–14 (cit. on p. 22).

[54] A. Iacovazzi, D. Frassinelli, and Y. Elovici. "The {DUSTER} attack: Tor onion service attribution based on flow watermarking with track hiding". In: *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. 2019, pp. 213–225 (cit. on p. 9).

[55] A. Iacovazzi, S. Sarda, and Y. Elovici. "Inflow: Inverse network flow watermarking for detecting hidden servers". In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE. 2018, pp. 747–755 (cit. on p. 9).

[56] R. Jansen, N. Hopper, and Y. Kim. "Recruiting new Tor relays with BRAIDS". In: *Proceedings of the 17th ACM conference on Computer and communications security*. 2010, pp. 319–328 (cit. on p. 11).

[57] W. Jia et al. "Voiceover: Censorship-Circumventing Protocol Tunnels with Generative Modeling". In: (2023) (cit. on pp. 23, 28).

[58] P. Jiang et al. "Poster: Metadata-private Messaging without Coordination". In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2023, pp. 3615–3617 (cit. on p. 17).

[59] M. Juarez et al. "A critical evaluation of website fingerprinting attacks". In: *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 2014, pp. 263–274 (cit. on p. 92).

[60] I. Karunanayake et al. "De-anonymisation attacks on Tor: A Survey". In: *IEEE Communications Surveys & Tutorials* 23.4 (2021), pp. 2324–2350 (cit. on p. 2).

[61] B. Kenig and T. Tassa. "A practical approximation algorithm for optimal k-anonymity". In: *Data Mining and Knowledge Discovery* 25 (2012), pp. 134–168 (cit. on p. 18).

[62] A. Kiran and N. Shirisha. "K-Anonymization approach for privacy preservation using data perturbation techniques in data mining". In: *Materials Today: Proceedings* 64 (2022), pp. 578–584 (cit. on p. 18).

[63] J. Knockel et al. "Every Rose Has Its Thorn: Censorship and Surveillance on Social Video Platforms in China". In: *5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 15)*. Washington, D.C.: USENIX Association, 2015-08. URL: https://www.usenix.org/conference/foci15/workshop-program/presentation/knockel (cit. on p. 1).

[64] A. Kwon, D. Lu, and S. Devadas. "XRD: Scalable messaging system with cryptographic privacy". In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 2020, pp. 759–776 (cit. on pp. 14, 16, 17).

[65] A. Kwon et al. "Atom: Horizontally scaling strong anonymity". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 406–422 (cit. on pp. 14, 16).

[66] A. W. de La Cadena Ramos. "Multipath Routing on Anonymous Communication Systems: Enhancing Privacy and Performance". PhD thesis. University of Luxembourg, Luxembourg, 2021 (cit. on pp. 11, 12).

[67] A. Langley et al. "The quic transport protocol: Design and internet-scale deployment". In: *Proceedings of the conference of the ACM special interest group on data communication*. 2017, pp. 183–196 (cit. on p. 22).

[68] D. Lazar, Y. Gilad, and N. Zeldovich. "Karaoke: Distributed private messaging immune to passive traffic analysis". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 711–725 (cit. on pp. 14, 16).

[69] S. Le Blond et al. "Herd: A scalable, traffic analysis resistant anonymity network for VoIP systems". In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 2015, pp. 639–652 (cit. on p. 1).

[70] S. Li, M. Schliep, and N. Hopper. "Facet: Streaming over videoconferencing for censorship circumvention". In: *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. 2014, pp. 163–172 (cit. on pp. 23, 28).

[71] T. Li et al. "HeteroTiC: A robust network flow watermarking based on heterogeneous time channels". In: *Computer Networks* 219 (2022), p. 109424 (cit. on p. 9).

[72] P. Liu, L. He, and Z. Li. "A Survey on Deep Learning for Website Fingerprinting Attacks and Defenses". In: *IEEE Access* 11 (2023), pp. 26033–26047. DOI: 10.1109/ACCESS.2023.3253559 (cit. on p. 10).

[73] Z. Liu et al. "DeepMetricCorr: Fast flow correlation for data center networks with deep metric learning". In: *Computer Networks* 233 (2023), p. 109904 (cit. on pp. 2, 9, 28).

[74] A. H. Lorimer, R. Jansen, and N. Feamster. "Traffic Splitting for Pluggable Transports". In: *Free and Open Communications on the Internet* (2024) (cit. on p. 11).

[75] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[76] S. Mansfield-Devine. "Tor under attack". In: *Computer Fraud & Security* 2014.8 (2014), pp. 15–18 (cit. on p. 2).

[77] A. Master and C. Garman. "A Worldwide View of Nation-state Internet Censorship". In: *Free and Open Communications on the Internet*. 2023. URL: https://www.petsymposium.org/foci/2023/foci-2023-0008.pdf (cit. on pp. 1, 28).

[78] N. Mathewson and R. Dingledine. "Practical traffic analysis: Extending and resisting statistical disclosure". In: *International Workshop on Privacy Enhancing Technologies*. Springer. 2004, pp. 17–34 (cit. on p. 9).

[79] R. McPherson, A. Houmansadr, and V. Shmatikov. "CovertCast". In: *Proceedings on Privacy Enhancing Technologies* 3 (2016), pp. 1–14 (cit. on pp. 23, 28).

[80] P. M. T. P. de Medeiros. "Distributed system for cooperative deanonymization of Tor circuits". In: (2021) (cit. on p. 2).

[81] U. Moeller. *Mixmaster Protocol Version 2*. Internet-Draft draft-sassaman-mixmaster-03. Work in Progress. Internet Engineering Task Force, 2004-12. 20 pp. URL: https://datatracker.ietf.org/doc/draft-sassaman-mixmaster/03/ (cit. on p. 5).

[82] H. Mohajeri Moghaddam et al. "Skypemorph: Protocol obfuscation for tor bridges". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 97–108 (cit. on p. 24).

[83] 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 (cit. on p. 11).

[84] 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 (cit. on p. 1).

[85] M. Nasr, A. Bahramali, and A. Houmansadr. "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 1962–1976. ISBN: 9781450356930. DOI: 10.1145/3243734.3243824. URL: https://doi.org/10.1145/3243734.3243824 (cit. on pp. 2, 9, 28).

[86] A. A. Niaki et al. "ICLab: A global, longitudinal internet censorship measurement platform". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 135–151 (cit. on pp. 1, 28).

[87] S. Nourin et al. "Measuring and Evading Turkmenistan's Internet Censorship: A Case Study in Large-Scale Measurements of a Low-Penetration Country". In: *Proceedings of the ACM Web Conference 2023*. WWW '23. Austin, TX, USA: Association for Computing Machinery, 2023, pp. 1969–1979. ISBN: 9781450394161. DOI: 10.1145/3543507.3583189. URL: https://doi.org/10.1145/3543507.3583189 (cit. on pp. 1, 28).

[88] V. Nunes et al. "Enhancing the Unlinkability of Circuit-Based Anonymous Communications with k-Funnels". In: *Proceedings of the ACM on Networking* 1.CoNEXT3 (2023), pp. 1–26 (cit. on pp. 8, 14, 15, 18, 28).

[89] S. E. Oh et al. "DeepCoFFEA: Improved flow correlation attacks on Tor via metric learning and amplification". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2022, pp. 1915–1932 (cit. on pp. 2, 9, 28).

[90] B. M. de Oliveira Carvalho. "Tirdeanon: Deanonymization and Traffic-Unobservability Analysis of Tir as a Strengthened Solution to Improve Anonymity Guarantees in Tor". Master's Thesis, Dep. Informática, FCT/UNL, 2023 (cit. on pp. 11, 20).

[91] E. Papadogiannaki and S. Ioannidis. "A survey on encrypted network traffic analysis applications, techniques, and countermeasures". In: *ACM Computing Surveys (CSUR)* 54.6 (2021), pp. 1–35 (cit. on p. 15).

[92] J. Pennekamp et al. "Multipathing traffic to reduce entry node exposure in onion routing". In: *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE. 2019, pp. 1–2 (cit. on pp. 2, 11).

[93] A. M. Piotrowska et al. "The loopix anonymity system". In: *26th usenix security symposium (usenix security 17)*. 2017, pp. 1199–1216 (cit. on pp. 1, 8, 11, 14–16, 28, 38).

[94] C. Rahalkar, A. Virgaonkar, and K. Varadan. *Analyzing Trends in Tor*. 2023. arXiv: 2208.11149 [cs.CR] (cit. on p. 2).

[95] R. Ramesh et al. "Network Responses to Russia's Invasion of Ukraine in 2022: A Cautionary Tale for Internet Freedom". In: *USENIX Security Symposium*. USENIX, 2023. URL: https://censoredplanet.org/assets/russia-ukraine-invasion.pdf (cit. on pp. 1, 28).

[96] F. Rezaei and A. Houmansadr. "FINN: Fingerprinting Network Flows Using Neural Networks". In: *Annual Computer Security Applications Conference*. ACSAC '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 1011–1024. ISBN: 9781450385794. DOI: 10.1145/3485832.3488010. URL: https://doi.org/10.1145/3485832.3488010 (cit. on pp. 2, 9).

[97] K. Sangeetha and K. Ravikumar. "A novel traffic dividing and scheduling mechanism for enhancing security and performance in the tor network". In: *Indian Journal of Science and Technology* (2015), pp. 689–694 (cit. on p. 11).

[98] S. Sasy and I. Goldberg. "SoK: Metadata-protecting communication systems". In: *Proceedings on Privacy Enhancing Technologies* (2024) (cit. on p. 17).

[99] H. Schaaf and S. Smith. *Go Report Card: A report card for your Go application*. [Online; accessed <today>]. 2015–. URL: https://www.goreportcard.com/ (cit. on p. 73).

[100] C. Scott, P. Wolfe, and M. Erwin. *Virtual private networks*. " O'Reilly Media, Inc.", 1999 (cit. on p. 28).

[101] A. Serjantov and P. Sewell. "Passive attack analysis for connection-based anonymity systems". In: *Computer Security–ESORICS 2003: 8th European Symposium on Research in Computer Security, Gjøvik, Norway, October 13-15, 2003. Proceedings 8*. Springer. 2003, pp. 116–131 (cit. on p. 6).

[102] M. Shen et al. "Subverting website fingerprinting defenses with robust traffic representation". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 607–624 (cit. on p. 10).

[103] Y. Shi and K. Matsuura. "Fingerprinting Attack on the Tor Anonymity System". In: *Information and Communications Security*. Ed. by S. Qing, C. J. Mitchell, and G. Wang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 425–438. ISBN: 978-3-642-11145-7 (cit. on p. 7).

[104] R. Shokri et al. "Unraveling an old cloak: k-anonymity for location privacy". In: *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*. 2010, pp. 115–118 (cit. on p. 18).

[105] P. Singh et al. *Connecting the dots in the sky: Website fingerprinting in low Earth Orbit Satellite internet*. 2024-03. URL: http://hdl.handle.net/10012/20389 (cit. on pp. 97, 100).

[106] P. Sirinam et al. "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: Association for Computing Machinery, 2018, pp. 1928–1943. ISBN: 9781450356930. DOI: 10.1145/3243734.3243768. URL: https://doi.org/10.1145/3243734.3243768 (cit. on p. 10).

[107] B. Sredojev, D. Samardzija, and D. Posarac. "WebRTC technology overview and signaling solution design and implementation". In: *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2015, pp. 1006–1009. DOI: 10.1109/MIPRO.2015.7160422 (cit. on p. 24).

[108] S. Staniford-Chen and L. T. Heberlein. "Holding intruders accountable on the internet". In: *Proceedings 1995 IEEE Symposium on Security and Privacy*. IEEE. 1995, pp. 39–49 (cit. on p. 9).

[109] Y. Sun et al. "{RAPTOR}: Routing attacks on privacy in tor". In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 271–286 (cit. on pp. 2, 10).

[110] L. Sweeney. "k-anonymity: A model for protecting privacy". In: *International journal of uncertainty, fuzziness and knowledge-based systems* 10.05 (2002), pp. 557–570 (cit. on pp. 18, 28).

[111] J. Teixeira and H. Domingos. "Strengthening of Tor Against Traffic Correlation with K-Anonymity Input Circuits". In: *Faculdade de Ciências e Tecnologia Universidade Nova de Lisboa* (2021) (cit. on pp. 18, 19, 28).

[112] N. Tyagi et al. "Stadium: A distributed metadata-private messaging system". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 423–440 (cit. on pp. 14, 16).

[113] J. Van Den Hooff et al. "Vuvuzela: Scalable private messaging resistant to traffic analysis". In: *Proceedings of the 25th Symposium on Operating Systems Principles*. 2015, pp. 137–152 (cit. on pp. 1, 14, 15).

[114] A. Vilalonga, J. S. Resende, and H. Domingos. "TorKameleon: Improving Tor's Censorship Resistance with K-anonymization and Media-based Covert Channels". In: *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. Los Alamitos, CA, USA: IEEE Computer Society, 2023-11, pp. 1490–1495. DOI: 10.1109/TrustCom60117.2023.00203. URL: https://doi.ieeecomputersociety.org/10.1109/TrustCom60117.2023.00203 (cit. on pp. 2, 3, 18, 20, 26, 28, 29, 52, 69, 84, 101, 103).

[115] A. Vilalonga, J. S. Resende, and H. Domingos. "Looking at the Clouds: Leveraging Pub/Sub Cloud Services for Censorship-Resistant Rendezvous Channels". In: *Free and Open Communications on the Internet* (2024) (cit. on p. 104).

[116] A. Vilalonga et al. "Comunicação: Algoritmos de Resposta Aleatória como Mecanismo para a Desvinculação entre o Tráfego de Entrada e de Saída em Sistemas de Anonimato Baseadas em Circuitos". In: *Atas do INForum 2024-15º Simpósio Nacional de Informática* (2024) (cit. on pp. 5, 7, 8).

[117] J. A. T. M. Vilalonga. "TorKameleon: Improving Tor's Censorship Resistance with K-Anonymization Media Morphing Covert Input Channels". Master's Thesis, Dep. Informática, FCT/UNL, 2022 (cit. on pp. 19, 20, 51).

[118] S. D. C. di Vimercati et al. "k-Anonymity: From Theory to Applications." In: *Trans. Data Priv.* 16.1 (2023), pp. 25–49 (cit. on p. 18).

[119] L. Wang et al. "Seeing through Network-Protocol Obfuscation". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA: Association for Computing Machinery, 2015, pp. 57–69. ISBN: 9781450338325. DOI: 10.1145/2810103.2813715. URL: https://doi.org/10.1145/2810103.2813715 (cit. on p. 2).

[120] M. Wang et al. "Leveraging strategic connection migration-powered traffic splitting for privacy". In: *arXiv preprint arXiv:2205.03326* (2022) (cit. on pp. 2, 11, 27, 29, 100).

[121] Q. Wang et al. "Censorspoofer: asymmetric communication using ip spoofing for censorship-resistant web browsing". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 121–132 (cit. on p. 24).

[122] S. Wang et al. "Look deep into the new deep network: a measurement study on the ZeroNet". In: *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part I 20*. Springer. 2020, pp. 595–608 (cit. on p. 1).

[123] T. Wang. "High precision open-world website fingerprinting". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 152–167 (cit. on p. 10).

[124] T. Wang et al. "Effective Attacks and Provable Defenses for Website Fingerprinting". In: *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014-08, pp. 143–157. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/wang_tao (cit. on p. 92).

[125] Z. Wang et al. "On how zero-knowledge proof blockchain mixers improve, and worsen user privacy". In: *Proceedings of the ACM Web Conference 2023*. 2023, pp. 2022–2032 (cit. on pp. 18, 28).

[126] Z. Wei et al. "Physical Layer Anonymous Precoding: The Path to Privacy-Preserving Communications". In: *IEEE Wireless Communications* 29.2 (2022), pp. 154–160. DOI: 10.1109/MWC.103.2100283 (cit. on p. 1).

[127] M. Wu et al. "How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic". In: *USENIX Security Symposium*. USENIX, 2023. URL: https://www.usenix.org/system/files/sec23fall-prepub-234-wu-mingshi.pdf (cit. on pp. 1, 28).

[128] D. Xue et al. "Fingerprinting Obfuscated Proxy Traffic with Encapsulated {TLS} Handshakes". In: *33rd USENIX Security Symposium (USENIX Security 24)*. 2024, pp. 2689–2706 (cit. on p. 22).

[129] L. Yang and F. Li. "mTor: A multipath Tor routing beyond bandwidth throttling". In: *2015 IEEE Conference on Communications and Network Security (CNS)*. 2015, pp. 479–487. DOI: 10.1109/CNS.2015.7346860 (cit. on p. 11).

[130] T. Ylonen and C. Lonvick. *The secure shell (SSH) protocol architecture*. Tech. rep. 2006 (cit. on p. 28).

[131] B. Zantout, R. Haraty, et al. "I2P data communication system". In: *Proceedings of ICN*. ICN, Springer Singapore. 2011, pp. 401–409 (cit. on pp. 1, 6).

[132] J. Zou et al. "SMRT: An Effective Malicious Node Resistance Design for Mixnets". In: *2022 7th IEEE International Conference on Data Science in Cyberspace (DSC)*. IEEE. 2022, pp. 110–117 (cit. on p. 13).