NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

GONÇALO GOMES RODRIGUES

BSc Degree in Informatics Engineering

THE UNOBSERVABILITY SCHEDULER

USE OF DIFFERENTIAL PRIVACY FOR UNOBSERVABLE PRIVACY-PRESERVED COMMUNICATION

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2024

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

THE UNOBSERVABILITY SCHEDULER

USE OF DIFFERENTIAL PRIVACY FOR UNOBSERVABLE PRIVACY-PRESERVED COMMUNICATION

GONÇALO GOMES RODRIGUES

BSc Degree in Informatics Engineering

Adviser: Henrique João
Associate Professor, NOVA University Lisbon

Examination Committee

Chair: Name of the committee chairperson
Full Professor, FCT-NOVA

Rapporteur: Name of a rapporteur
Associate Professor, Another University

Members: Another member of the committee
Full Professor, Another University

Yet another member of the committee
Assistant Professor, Another University

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
February, 2024

The Unobservability Scheduler
Use of Differential Privacy for Unobservable Privacy-Preserved Communication

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.29) [38].

Abstract

The internet has become an indispensable tool for accessing information and exercising freedom of speech, a fundamental right that empowers individuals worldwide. However, authoritarian regimes and government agencies tend to suppress this right through surveillance and censorship activities. These efforts have become increasingly sophisticated, employing techniques such as traffic analysis attacks and advanced traffic correlation methods to monitor and undermine user anonymity.

In response to the growing demand for secure and private communication, Anonymity Networks like Tor have emerged as essential tools for safeguarding online end-to-end communication and fighting censorship, which millions of people rely on daily. However, recent research demonstrated that potential de-anonymization attacks can be performed by powerful state-level adversaries by traffic correlation and fingerprinting using advanced deep machine learning techniques.

This dissertation addresses these challenges by strengthening Tor network's defenses against the above-mentioned attacks. The main goal is to develop an innovative solution targeted to improve Tor relay nodes leveraging the existing software architecture by introducing formal privacy guarantees based on differentially private circuits. We expect that our proposed solution will maintain practical levels of throughput and latency for end-to-end communication, ensuring compatibility with real-world usage scenarios while providing users with adaptable and more robust anonymity protection.

As stated above, the core of the proposed solution is the integration of Differential Privacy principles to dynamically introduce carefully bounded random noise into traffic flows, mitigating correlation attacks and empowering a better solution resilient against fingerprinting. To the best of our knowledge, the expected dissertation contributions can represent a pioneering effort in incorporating Differential Privacy into the software architecture of Tor relay nodes. Moreover, the use of Differential Privacy will allow for a formally proven, scalable, and effective mechanism that bolsters online anonymity, resistant against state-of-the-art internet censorship techniques and upholding the fundamental right to free expression in an increasingly monitored digital world.

Keywords: Anonymity Networks, Differential Privacy, End-to-End Privacy Enhanced Guarantees, Privacy-Preserving Communication, Traffic Analysis Attacks, Traffic Correlation

Resumo

A Internet tornou-se uma ferramenta indispensável para aceder à informação e exercer a liberdade de expressão, um direito fundamental que confere poder aos indivíduos em todo o mundo. No entanto, os regimes autoritários e as entidades governamentais tendem a restringir este direito através de práticas de vigilância e censura. Estes esforços têm-se tornado cada vez mais sofisticados, recorrendo a técnicas como a análise de tráfego e métodos avançados de correlação de dados para monitorizar e comprometer o anonimato dos utilizadores.

Em resposta à crescente necessidade de comunicações seguras e privadas, surgiram redes de anonimato como o Tor, que desempenham um papel essencial na proteção das comunicações online e na resistência à censura, das quais milhões de pessoas dependem diariamente. No entanto, investigações recentes demonstraram que ataques de desanonimização podem ser conduzidos por adversários poderosos a nível estatal, utilizando correlação de tráfego e técnicas avançadas de aprendizagem automática para identificar padrões e comprometer a privacidade dos utilizadores.

Esta dissertação aborda estes desafios, reforçando as defesas da rede Tor contra os ataques supramencionados. O principal objetivo é desenvolver uma solução inovadora para melhorar os nós de retransmissão Tor (Relay Nodes), aproveitando a arquitetura de software existente e introduzindo garantias formais de privacidade baseadas em circuitos privados diferenciados. Pretende-se que a solução proposta preserve níveis práticos de throughput e latência para comunicações de ponta a ponta, garantindo a compatibilidade com cenários de utilização real e proporcionando aos utilizadores uma proteção do anonimato mais robusta e adaptável.

Como referido anteriormente, o cerne da solução proposta assenta na integração dos princípios da Privacidade Diferencial, introduzindo dinamicamente ruído aleatório cuidadosamente delimitado nos fluxos de tráfego, mitigando os ataques de correlação e reforçando a resistência à recolha de impressões digitais. Tanto quanto é do nosso conhecimento, os contributos esperados desta dissertação poderão representar um avanço pioneiro na incorporação da Privacidade Diferencial na arquitetura de software dos nós de retransmissão Tor (Relay Nodes). Além disso, a aplicação da Privacidade Diferencial

permitirá o desenvolvimento de um mecanismo formalmente comprovado, escalável e eficaz, que reforça o anonimato online e se mantém resiliente face às técnicas mais avançadas de censura na Internet, protegendo, assim, o direito fundamental à liberdade de expressão num mundo digital cada vez mais sujeito a monitorização.

Palavras-chave: Redes de Anonimato, Privacidade Diferencial, Comunicação Preservadora de Privacidade, Ataques de Análise de Tráfego, Garantias Reforçadas de Privacidade Ponto a Ponto, Correlação de Tráfego

Contents

List of Figures

Listings

1

Introduction

Freedom of speech and access to information are fundamental rights that many people take for granted. However, in some regions of the world, these rights are denied due to powerful and oppressive regimes and state-level adversaries with access to vast resources that control the flow of information and restrict the access to the internet, in order to control the population [3, 73, 2].

To protect these populations and ensure their access to freedom of speech and information, 'anonymity networks' are systems designed to preserve user anonymity while protecting their privacy and security. As authoritarian regimes intensify their efforts to restrict and control their population access by employing advanced techniques, these networks are becoming increasingly important to protect the privacy and security of users, as well as the privacy and anonymity guarantees offered by these networks. Consequently, these networks are becoming increasingly important to protect the privacy and security of users, as well as the privacy and anonymity guarantees offered by these networks.

These regimes resort to the use of traffic analysis techniques to passively monitor the traffic and infer information about the users, in order to enforce and control the network and their users. Traffic analysis attacks are a significant threat to anonymity networks, as they can be used to infer users' activities and deny their privacy, as well as de-anonymize them [8, 69, 31]. Researchers have demonstrated that advances in machine learning lead to more sophisticated and effective traffic correlation attacks [42, 25, 7], such as DeepCoFFEA [45] and FINN [52], which leverage deep learning techniques to correlate traffic patterns and infer users' activities with high accuracy, in anonymity networks like Tor [34]. Fingerprinting attacks also represent a major threat, as they enable attackers to infer information about users' online activities—even in encrypted traffic—by eavesdropping and collecting side-channel information between the user and the entry node. Using this information, an attacker can apply machine learning techniques to infer a user's online activities—such as visited websites—by analyzing traffic patterns with high accuracy [57, 51, 10].

As attacks grow more sophisticated, journalists, whistleblowers, and anyone striving to exercise their freedom of speech in oppressive regions face increasing risks of identification

and persecution. This not only diminishes the utility of anonymity networks but also fuels mistrust and fear. To combat these threats, the scientific community must develop robust solutions that strengthen anonymity guarantees, backed by proven privacy and security measures.

1.1 Motivation

Tor [15], one of the most popular anonymity networks, is an anonymous communication service based on the Onion Routing Protocol. This network is designed to protect users' privacy and security by routing their traffic through a network of volunteer-operated servers, encrypting it at each step, and ensuring that no single server knows both the source and destination of the traffic.

As mentioned earlier, anonymity networks remain vulnerable to attacks such as traffic analysis, which threatens the anonymity and security of the users these networks are designed to protect [8, 69, 31, 13, 39, 57, 51, 10, 56, 72]. To address the challenge of circumventing censorship, researchers have proposed a range of solutions.

Some of these proposed sophisticated methods include Traffic Encapsulation, Pluggable Transports, k-Anonymity, Multipath Strategies, and Traffic Splitting. Traffic encapsulation is a method that conceals data by embedding packets within other data formats or protocols, thereby obfuscating their true content. This technique leverages protocols, such as those for web browsing and video streaming, to disguise network traffic, making it indistinguishable [64, 47, 68, 5, 20, 26]. Pluggable Transports serve as a mechanism for obfuscating network traffic by employing diverse protocols to connect to the Tor network, thereby complicating an attacker's ability to identify and analyze the traffic [50, 49, 64].@k-Anonymity is a formal privacy definition designed to ensure that an individual's data remains indistinguishable within a larger set [43, 60, 24]. Meanwhile, Multipath Strategies and Traffic Splitting enhance security by distributing data across multiple concurrent paths, reducing the risk of interception and traffic analysis [64, 47, 48, 66]. Similarly, traffic shuffling is a technique that focus on reordering packets, with statistical or cryptographic techniques, to make it difficult for an attacker to correlate the traffic [54, 36, 11, 48, 70].

Some of these solutions have presented promising results in terms of privacy and security. However, we argue that the privacy guarantees provided by these solutions are not quantifiable, potentially making it difficult to evaluate their effectiveness and to compare them with other solutions. To address this problem, we apply Differential Privacy (DP) [16, 18, 43] to the Tor network in order to enhance its resistance against fingerprinting and correlation attacks, thereby provide quantifiable formal privacy guarantees to users, while maintaining practical levels of performance. DP was originally designed for database analysis and privacy preserving analysis but some works on communications systems have shown that DP can also be used to enhance privacy [48, 63, 65, 71, 53].

1.2 Problem

Although anonymity networks are designed to provide robust privacy and security, advancements in traffic analysis and correlation techniques have increasingly exposed vulnerabilities, reducing their reliability and trustworthiness for users — particularly in regions where pervasive surveillance by authoritarian regimes is a critical concern. These regimes actively invest in sophisticated surveillance methods and traffic manipulation strategies to maintain control over digital communication [3, 73, 2]. Tor [15], despite being one of the most widely used anonymity networks, is not immune to these challenges. It remains vulnerable to several attacks, including traffic correlation, congestion-based attacks, timing-based correlations, and fingerprinting techniques [8, 34, 69, 31]. Such vulnerabilities highlight the ongoing arms race between those striving to safeguard online anonymity and those seeking to undermine it through technical and infrastructural advancements, such as deep learning techniques.

The absence of standardized metrics for evaluating privacy guarantees in a formal and rigorous manner raises concerns about the effectiveness of anonymizing networks like Tor. Research has demonstrated vulnerabilities in the Tor network to various attacks, including traffic analysis. For instance, Chakravarty et al. showed that statistical correlation techniques can be used to perform successful traffic analysis attacks [8]. Additionally, Shi and Matsuura demonstrated that user privacy can be compromised through fingerprinting attacks [55].

1.3 Goals

As stated previously in Section 1.1, it is important to ensure that anonymity networks are secure against threats like traffic fingerprinting and correlation attacks, empowered by deep learning techniques, and to address the lack of standardized privacy metrics to quantify the anonymity guarantees provided by these networks.

To address this challenge, we incorporate Differential Privacy (DP) into the Tor open-source project, by integrating Differential Privacy-based mechanisms in the existing software architecture of Tor relay node, in order to dynamically inject carefully bounded random noise into traffic flows to provide formal privacy guarantees to users. Our work builds upon the foundation laid by Jansen et al. [32], who introduced KIST, a congestion control scheduler for Tor designed to enhance network performance by reducing latency and increasing throughput. While KIST achieves significant efficiency gains, its focus remains solely on performance, without introducing additional privacy guarantees or defenses against traffic analysis attacks beyond those inherent to Tor. Inspired by this limitation, we extend KIST's approach, not only preserving its performance benefits but also reinforcing privacy protections and bolstering resistance to traffic analysis attacks.

In this manner, we reinforce the Tor end-to-end communication anonymity guarantees, even under traffic fingerprinting and correlation attacks. To ensure real-world applicability

of this solution, it is important to maintain practical levels of throughput and latency for end-to-end communication.

1.3.1 Expected Contributions

To address the stated goal, we made the following contributions[1]:

1. Definition and formalization of the system design model to incorporate differentially private-based circuits as a possible interesting solution for Tor relay nodes to improve Tor anonymity conditions against traffic fingerprinting and correlation attacks;

2. Specification of the software architecture and modular components for the engineering effort using Differential Privacy-based mechanisms and related components in extended Tor nodes;

3. Prototyping of the proposed solution where the implemented nodes were used in a validation test bench and publishing the prototype in open-source for the possible use of interested researchers and practitioners;

4. Validation of the designed solution and prototype considering three main pillars: (*i*) Development of formal security proofs behind the designed solution; (*ii*) conduct an extensive experimental on performance indicators including communication latency, throughput, as well as resources' usage for running the proposed nodes, and (*iii*) experimental observation of the unobservability properties of the designed solution against traffic correlation and fingerprinting techniques.

In the final validation effort, we obtained comparative metrics using a test bench that compared our solution with the existing Tor network and relay nodes.

1.4 Report Organization

The remainder of this report is organized as follows: Chapter 2 introduces anonymity networks, Tor and Differential Privacy, as well as some other related topics. The proposal of work is presented on Chapter 3, which provides some system model and guidelines for the development and discusses the proof method to verify the work. Then, we explore the implementation details, techniques and technologies used to develop the proposed solution in Chapter 4. Chapter 5 presents the evaluation of the proposed solution, including the experimental setup, performance evaluation, and security analysis. Finally, we conclude the report in Chapter 6, summarizing the main findings and contributions of this work, and suggesting directions for future research.

[1]The dissertation effort was planned as a research project task within NOVA LINCS (Nova Laboratory of Informatics and Computer Science), Computer Systems Research Group. The elaboration phase involved integrating the expected contributions with collaborative engagement and follow-up of two PhD students: João Afonso Vilalonga and Hugo Gamaliel Pereira.

2

Related Work

In this chapter, we provide a more detailed overview of the state-of-the-art concepts introduced in the previous chapter. First, we discuss the concept of anonymity networks and Mixnets, followed by a detailed review of notable works in this field. Next, we examine the Tor network, including key technologies such as Pluggable Transports and the Tor relay node scheduler. We then introduce *k*-Anonymity, and finally, we provide an in-depth overview of Differential Privacy.

2.1 Anonymity Networks & Mixnets

Mix Networks (Mixnets) were introduced by Chaum [9] as a public key cryptography based protocol designed to obscure the relationship between senders and receivers in a communication network, as well as its content, without the need for a universal trusted authority. Mixnets consist of a series of intermediary servers, called 'mixes', through which messages are decrypted, encrypted, randomly permuted and sent forward. To transmit messages, participants encapsulate their data within successive layers of encryption, with each layer corresponding to a specific mix in the network — a method conceptually akin to onion routing [21]. The number of encryption layers, and thus the length of the messages, is proportional to the number of mixes. Upon receiving a message, each mix removes the outermost layer of encryption, processes the decrypted instructions, and forwards the partially decrypted message to the subsequent mix. This iterative procedure continues until the message arrives at the final mix. The final mix then removes the last layer of encryption and transmits the plaintext message to the intended recipient. Intrinsically, this system ensures that no single mix possesses complete knowledge of both the sender's identity and the recipient's address, thereby achieving a robust level of anonymity.

Various designs of mixnets have been proposed, based on Chaum's work, addressing multiple security and performance issues. Flash Mixing, introduced by Jakobsson [28], focuses on achieving strong anonymity guarantees with reduced latency and computational overhead by shuffling messages, broadcasting the encrypted list to all mixes, who then together compute the output. Hybrid Mixnets [29, 46] efficiently combine public-key

and symmetric-key cryptography. Real-time Mixnets [33] aim voice and data communication where continuous data streams have to be transmitted and provide low-latency communication, as long as a certain delay at the start of a connection is tolerable. Work in this field has focused both on security and privacy guarantees, as well as on performance and scalability, with the goal of providing a practical and efficient solution for anonymous communication. Mixmaster [41] allows the sender of a message to remain anonymous to the recipient. Mixminion [12] uses fixed sized messages and supports anonymous replies and ensures forward anonymity using link encryption between nodes. Onion Routing [21] followed a similar approach, but focused on low latency communication, where messages are encrypted in layers and decrypted by a chain of authorized nodes.

Tor [15] is one of the most popular low latency anonymity system and protects against sender-receiver message linking against a partially global adversary and ensures perfect forward secrecy, messages' integrity, and congestion control. However, Tor is vulnerable to traffic analysis attacks, if the adversary is able to observe the entry and exit points of the network. Mix-In-Place (MIP) [44] is a mixnet that uses a cascade of functions in a single proxy, instead of multiple intermediary nodes, to provide anonymity, and proving more resistant to traffic analysis attacks. Vuvuzela [23] operates in rounds, leading to offline users' inability to receive messages and all messages must transverse a single chain of relay servers. This design protects against both active and passive adversaries unless there is no honest mix node in the network.

2.1.1 Groove

Groove [4] is a scalable, metadata-private messaging system designed to support users with multiple devices, enabling them to send messages at any time, even when recipients are offline, while conserving bandwidth and energy. Built on mixnets, Groove ensures unlinkability between senders and their messages by shuffling batches of messages across servers. Traditional mixnets, however, require all users to submit messages during every round and receive messages at a synchronized rate to prevent correlated traffic patterns, which limits their scalability and usability. To overcome these limitations, Groove introduces a novel approach called oblivious delegation. In this model, users interact with an untrusted service provider that participates in the mixnet on their behalf and synchronizes their clients across all devices. This ensures that even if the service provider is compromised, an adversary cannot infer communication metadata. Unlike prior systems such as Karaoke [36], Stadium [63], and Vuvuzela [23], which require users to be online simultaneously, communicate in synchronized rounds, and support only single-device usage, Groove addresses these restrictions, offering asynchronous messaging and multi-device support. A significant advantage of Groove lies in its ability to minimize the resource demands of each message channel on the mixnet, with a particularly notable reduction in memory usage compared to earlier systems. Groove ensures Differential Privacy even against attackers with full network control and the ability to compromise

multiple servers. Users communicate through persistent message channels, similar to Tor's circuits, while their service provider handles message submission to the mixnet, stores received messages, and synchronizes clients across devices. To maintain anonymity, Groove leverages Parallel Mixnets, which efficiently scale with the number of servers by offering multiple parallel routes for processing messages. Differential privacy is a core goal of Groove, ensuring that traffic patterns between a user's client and their service provider do not disclose information about the user's communication partners. This is achieved through a scheduler that operates independently of the sender-recipient relationship. In contrast to Loopix [48], Groove maintains user privacy even if their service provider is compromised, eliminating the need for users to operate their own servers.

2.1.2 Loopix

Loopix [48] is a low-latency anonymous communication system offering bidirectional 'third-party' sender and receiver anonymity as well as unobservability. It employs cover traffic and Poisson mixing to provide strong anonymity guarantees and resist traffic analysis by a Global Network Adversary (GNA). The system ensures robust sender-receiver unlinkability against a Global Passive Attacker (GPA) capable of observing all network traffic between users, providers, and mix servers, even in cases where some mix nodes are compromised. Loopix also guarantees sender online unobservability under a corrupt provider and receiver unobservability under the condition of an honest provider. Consequently, the GPA cannot infer the type of transmitted messages, while intermediate nodes remain unable to distinguish between real messages, dropped cover messages, or loops of traffic generated by clients and other nodes. To protect against active attacks, mixes and clients employ self-monitoring mechanisms through self-injected traffic loops. These loops not only act as cover traffic to strengthen anonymity but also enhance sender and receiver unobservability. Similar to Groove, Loopix utilizes service providers to mediate access to the network, manage accounting, and enable offline message reception. These semi-trusted providers enforce rate limits, store messages for offline users, and facilitate message retrieval at a later time. Despite this reliance, Loopix is resilient against adversaries capable of observing all communications and conducting active attacks. A distinguishing feature of Loopix is its continuous operation, unlike Groove's deterministic round-based approach. Messages in Loopix can be retrieved at any time, ensuring users do not lose messages while offline. Furthermore, Loopix employs Poisson mixing, a simplified version of the stop-and-go mixing strategy [35], which introduces independent delays for each message, making packet timings unlinkable. Unlike circuit-based onion routing, where a fixed path is established, Loopix determines the communication path for each individual message independently, even when sent between the same pair of users. This approach enhances privacy and unpredictability. The Poisson mix operates as follows: mix servers listen for incoming packets, check for duplicates, and decode received messages using their private keys. Duplicates are discarded, and the next mix

packet is extracted. Decoded packets are not forwarded immediately; instead, each packet is delayed based on a pre-determined delay specified by the source.

2.1.3 Stadium

Stadium [63] is a point-to-point messaging system that ensures metadata and data privacy while efficiently scaling its workload across hundreds of low-cost providers operated by different organizations. Stadium achieves its provable privacy guarantees through the use of noisy cover traffic and Differential Privacy, which bounds metadata leakage over time. The system employs a mixnet architecture, where user messages and noise messages are distributed among providers and mixed in parallel. Each provider processes only a fraction of the total messages, ensuring scalability and efficiency. As long as at least one provider remains honest, Stadium achieves global verification and secure mixing. Communication in Stadium occurs in fixed rounds, with each round processing a batch of messages accumulated from users during the preceding interval. To enforce Differential Privacy, servers generate cover traffic at the start of each round. Noise messages are injected into the system in a single, large step before mixing begins. However, this design leaves room for malicious servers to discard noise messages before mixing. To address this vulnerability and ensure that noise remains in the system, Stadium employs cryptographic techniques for verifiable message processing. These techniques enable honest servers to verify the actions of others, preserving the system's Differential Privacy guarantees. To optimize the computational workload of this verification process, Stadium introduces hybrid verifiable shuffling. Stadium's mixing process relies on a parallel mixing scheme. Each server initially processes a small fraction of input messages, mixes them, and redistributes the mixed messages among other servers. This cycle of mixing and redistribution is repeated multiple times. Although messages are initially partitioned based on their originating server, the repeated cycles of splitting and mixing result in global mixing across all servers.

2.1.4 MIRACE

MIRACE [47] is a censorship circumvention system developed by Pereira that leverages dynamically constructed circuits to provide secure communications. This work allows traffic to be split across multiple circuits, each composes of several nodes. These solutions combine traffic splitting with encapsulations, through the combination of TLS, QUIC and WebRTC tunnels for diverse covert communication strategies, and traffic shaping, by the addition of jitter and padding. Pereira demonstrated that MIRACE is able to ensure secure communications against correlation attacks, even with machine learning-based website fingerprinting attacks. The author also showed that MIRACE is able to maintain practical levels of latency and throughput, even with the addition of nodes. Under fingerprinting attacks, MIRACE showed strong levels of resistance with high accuracy and the ability to effectively obscure traffic patterns.

2.1.5 Problems & Analysis

Mixnets have been analyzed in terms of security and privacy guarantees, as well as performance and scalability. Zhu et al. revealed that flow-correlation attacks can be used to de-anonymize users in mixnets, in relation to the system parameterization, such as sample size, noise level, payload flow rate, and detection rate [72]. Shmatikov and Wang demonstrated that Mixnets are vulnerable to timing analysis, even with defenses in place, by the use of advanced statistical techniques, such as cross-correlation and machine learning, successfully de-anonymizing users [56].

2.2 Tor network

Tor [15] is a circuit-based low-latency anonymous communication service based on *onion routing* that aims to anonymize TCP-based applications, such as web browsing, SSH and instant messaging. The network relies on *Onion Routers* (ORs) that are responsible for maintaining TLS connections to every other Onion Router and for forwarding traffic along the circuits, and are also maintained by volunteers.

An *Onion Proxy* (OP) is the local software run on the client and that handles connections with the users' applications, fetches the current network information and the lists of *Onion Routers* (ORs) and builds circuits through the network. Each proxy maintains TLS connections to nodes they have been in contact recently and a pair of keys: long-term identity key and short-term onion key. Long-term keys are used to sign TLS certificates and its router descriptor. Short-term keys are used to decrypt requests, negotiate ephemeral keys and set up circuits.

Traffic travels the network through 512 bytes fixed-size *cells* with a header and a payload. The header contains the circuit identifier and the command to describe what to do with the payload. Based on the header's command, cells are either control cells or relay cells. Control cells are used to manage circuits and connections, while relay cells are used to carry data and can only be sent after a circuit is established. Relay cells have a digest assigned by the sender and its header and payload are iteratively encrypted with the symmetric key of each hop up to the target Onion Router. Thus, only the last router in the circuit can read the payload, given the fact that the digest is encrypted to a different value at each step.

The sequences of ORs which traffic is routed through are called *circuits*, normally composed by 3 relays (entry or guard, middle and exit). To create a new circuit, a user's Onion Proxy must execute a Diffie-Hellman key exchange with each OR in the circuit, one hop at a time. Hence, each Onion Router knows the previous and the next node in the circuit, but not the source and destination of the data. Once the circuit is established, relay cells are sent through the circuit,

2.2.1 Tor Bridges & Pluggable Transport

One vulnerability about Tor is the traffic blocking by ISPs of censored regimes. One way to perform this blocking is by blacklisting Tor relays, since the list of their IP addresses is public, which prevents clients from establishing circuits. Bridges come as a solution to this problem, where unpublished proxies forward client's traffic to a Tor entry relay, making the client connect to some unlisted bridge, rather than some potentially blacklisted relay [40]. Nevertheless, bridges are still vulnerable to traffic fingerprinting attacks. Matic, Troncoso, and Caballero demonstrated that 55% of public bridges are vulnerable to the traffic blocking referred before. Even with encrypted traffic, Tor's traffic is still identifiable by its packet size, currently fixed-sized 512 byte cells, and by the TLS byte patterns, more precisely the TLS extension called Server Name Indication (SNI), which adds the domain name to the TLS header without any encryption.

To overcome this issue, Tor bridges support *pluggable transports* (PT) [50] which transform the Tor traffic flow between the client and the bridge, making traffic monitoring between these look innocent, instead of actual Tor traffic. Some examples of PTs are *obfs4*, *meek*, *Snowflake* and *WebTunnel* [49]. *obfs4* obfuscates Tor traffic to make it appear random, thwarting efforts by censors to detect it through Internet scanning; *meek* disguises Tor traffic to resemble ordinary browsing activity on prominent websites; *Snowflake* routes your connection through volunteer-operated proxies to make it look like you're placing a video call instead of using Tor; and *WebTunnel* camouflages Tor traffic by making it indistinguishable from standard HTTPS requests, thereby creating the illusion of accessing a secure website.

2.2.2 Tor Hidden Services

Tor circuits, by their inherent design, do not ensure the anonymity of the receiver. This limitation arises from the fact that, for a client to reach a destination via an exit node, the recipient's IP address must be publicly accessible. To address this vulnerability, Tor introduces hidden services and the concept of a rendezvous point (RP) — a Tor relay node that facilitates recipient anonymity through the construction of dual circuits.

Hidden services enable any entity to host a TCP-based service without disclosing its IP address. Clients wishing to access such services utilize a special onion service address, which is derived from the service's public key. This cryptographically generated address provides a secure means of routing without exposing network-layer identifiers.

The fundamental mechanism underpinning hidden services involve the establishment of two distinct Tor circuits: one from the client to the rendezvous point and another from the hidden service to the same point. These circuits are constructed independently, and neither endpoint is aware of the other's IP address. The rendezvous point serves solely as an intermediary to facilitate communication between the two parties.

This architecture inherits the sender anonymity traditionally offered by Tor circuits and extends it to receivers. Only the respective entry nodes of the circuits are aware of the

origin of the traffic, and no single entity can link the sender and receiver. Notably, the exit node of the client circuit terminates at the rendezvous point, perceiving it as the ultimate recipient. Consequently, the client does not require knowledge of the hidden service's IP address, thereby ensuring the recipient's anonymity.

In summary, the use of rendezvous points and independently constructed circuits within Tor's hidden service framework effectively achieves mutual anonymity, safeguarding both client and service identities during communication.

2.2.3 Tor Circuit Scheduling

Traffic handling in Tor involves several buffers and schedulers. Each circuit is used by only one client but if multiple circuits use the same two ORs, they share the same connection. That is, OR will have multiple simultaneously connections with other ORs but only one to any given OR, and each connection will transport data for various circuits. When a OR receives a packet from an TCP stream, the packet gets demultiplexed, together with other possible incoming packets from different TCP streams, and they get placed into the kernel socket input buffers. Here packets are processed by the OS, usually in FIFO order, and sent to the Tor input buffers. Upon receiving the packets, Tor will remove the TLS layer and onion-encrypted (or onion-decripted, depending on the circuit's direction) and finally enqueued in the Tor Circuit Queue. Each relay maintains a queue for each circuit that it serves. Cells from the same Tor input buffer might not be enqueued in the same circuit queue, as they might belong to different circuits. The cells are then selected, dequeued, onion-encrypted and stored in a Tor output buffer. The data is then written to a kernel socket output buffer when the Tor output buffer contains sufficient data to form a TLS packet.

Tor has a congestion control mechanism (Circuit-Level Throttling), designed to regulate communications across circuits, which ensures resource distribution, prevents network congestion and provides safety against attacks such as Denial-of-Service. As Tor is a low latency network and its security guarantees rely on how many users are using the network, it is important to have a good congestion control mechanism and practical levels of latency and throughput. However, it has been shown that Tor's Circuit Scheduling Algorithm allows busy circuits to crowd out bursty circuits, leading to congestion and latency issues [62]. Additionally, Jansen et al. have demonstrated that Tor's congestion occurs in the kernel socket buffers. Both these works have proposed solutions to improve Tor's congestion control and latency issues [32].

Even with the improvements in congestion control, as merging Tang and Goldberg and Jansen et al. works, Tor is still vulnerable to traffic analysis attacks such as website fingerprinting and correlation-based attacks [32]. Even though none of these works presented additional security risks or vulnerabilities to the Tor network, they have shown little or no improvement in the network against these attacks.

2.2.3.1 Exponential Weighted Moving Average (EWMA)

Tang and Goldberg proposed a circuit scheduling algorithm that handles circuits by their recent activity, where bursty circuits have higher priority over busy ones [62]. Generally, bursty circuits are those used for web browsing and instant messaging, and also referred as interactive streams. On the other hand, busy circuits are those used for file transfers, and also referred as non-interactive streams. Interactive streams are more sensitive to latency while non-interactive streams tolerate higher delays. The approach is based on the Exponential Weighted Moving Average (EWMA) algorithm, which assigns a weight to each circuit based on the number of cells sent on each circuit. When selecting the circuit to process, the algorithm chooses the one with lowest EWMA value. Usually, newly created circuits have a lower EWMA value, leading to a higher priority. Even though this algorithm has merged into Tor, it has been demonstrated that it actually reduces performance for clients under some network conditions [30]. Additionally, this work does not address the adversaries' ability to correlate and fingerprint users' traffic, which is a major issue in Tor network.

2.2.3.2 Kernel Informed Socket Transport (KIST)

Motivated by the still congestion problem with Tor and the lack of understanding of where this problem occurs, Jansen et al. found that the congestions occurs inside the kernel socket buffers and on Tor's sockets management. They proposed a new scheduling algorithm, Kernel Informed Socket Transport (KIST), that solves these problems by choosing from all circuits with writable data rather than just those belonging to a single TCP socket and by dynamically managing the amount of data written to each socket based on real-time kernel and TCP state information that can be queried from user space. Consequently, KIST reduces Tor's circuit congestion by more than 30%, reduces network latency by 18%, and increases network throughput by nearly 10% [32]. KIST was merged and configured as default socket scheduling algorithm in Tor since January 2018, replacing the EWMA-based algorithm, referred in Section 2.2.3.1. Although, KIST authors acknowledge that it does not affect the adversaries abilities to collect accurate measurements required for the throughput correlation attack, compared to the 'vanilla' Tor scheduling algorithm.

2.2.4 Tor Vulnerabilities

Studies have pointed that Tor is susceptible to traffic analysis attacks, where an adversary can infer information about a user, like who is communicating with whom, by observing the traffic patterns. Here we cover some of the most common traffic analysis attacks against Tor, which have been demonstrated to be effective in de-anonymizing users.

2.2.4.1 Website Fingerprinting Attack

Chakravarty et al. also proposed a traffic analysis attack model against Tor where the adversary uses statistical correlation over the server to exit and entry to client traffic to find similar traffic patterns [8]. This attack consists on clients generating sustained traffic for a long period of time, by downloading a file for example, and the adversary being therefore able to inject traffic patterns by perturbing the TCP connection. Thus, the adversary can obtain traffic analysis from the server to exit node and correlate it with the traffic from the entry node to the client, by finding the flow which carries the injected fingerprint. Testing showed that the attack was able to correctly identify the source of anonymous traffic 81.4% of the time. Deep Fingerprinting [57] is a website fingerprinting designed using deep learning techniques, that uses a simple input format and does not require handcrafting feature for classification. The authors demonstrated that this attack is 98.3% accurate in identifying the website visited by the user. Sirinam et al. proposed a website fingerprinting attack that uses triplet networks and a machine learning technique, that requires few training samples, called N-shot learning, which reduces the effort of gathering and training with large datasets [58]. Finally, this work achieved up to 95% accuracy in identifying the website visited by the user, only by using 20 examples per website.

2.2.4.2 Correlation-Based Attacks

Correlation attacks were acknowledged by Dingledine, Mathewson, and Syverson [15] as a potential threat to users' anonymity guarantees in the Tor network. These attacks exploit the adversary's ability to observe both the entry and exit points of the network, allowing them to correlate the traffic patterns and de-anonymize users. Sun et al. [59] proposed asymmetric traffic analysis as an end-to-end timing analysis that allows AS-level adversaries to compromise Tor users' anonymity. Internet path from exit relays to the web server may differ from the path from the web server to the exit relay, allowing the adversary to observe the TCP acknowledgement traffic on the path from the server to the exit relay, even if the adversary is enabled to observe the data traffic on path from the exit relay to the server. This attacks might be applicable where the adversary observes: data traffic from the client to the entry relay and from the exit relay and the server; data traffic from the client to the entry relay and TCP acknowledgement traffic from the server to the exit relay; TCP acknowledgement traffic from the entry relay to the client and data traffic from exit relay to the server; or TCP acknowledgement traffic from the entry relay to the client and TCP acknowledgement traffic from the server to the exit relay. The authors proposed a suite of new attacks against Tor called 'RAPTOR', where they demonstrated an accuracy of 95% in correlating client/server pair.

DeepCorr [42] and its extension DeepCoFFEA [45] are both systems that leverage advanced deep learning techniques to correlate traffic. DeepCorr is able to correlate Tor flows by first using deep learning to learn a correlation function and then using this

function to cross-correlate the traffic. In contrast to website fingerprinting, this work has no need to learn target destinations, instead the function can be used to link flows on arbitrary destinations. Additionally, it does not require the traffic to be sent in the same circuits used to learn the function. DeepCoFFEA extends DeepCorr by using a modified triplet network approach and amplification techniques, demonstrating that it achieved lower computational costs and higher efficiency compared to DeepCorr, improving the accuracy of the correlation attack.

2.3 k-Anonymity

To address the exponential growth in the number and variety of data collections containing person-specific information, and the pressing need to prevent privacy compromise, k-Anonymity [60] emerged as a foundational privacy model. This model ensures data anonymity not only by removing explicit identifiers such as names, addresses, or phone numbers but also by protecting against re-identification through linking or matching data with external sources. It achieves this by mitigating the risk posed by unique characteristics within the dataset.

Earlier work by Sweeney demonstrated the vulnerability of anonymized data to linkage attacks using the 1990 U.S. Census. They highlighted that 87% of the U.S. population reported characteristics that rendered them unique, and over half of the population could be uniquely identified using just three attributes: place, gender, and date of birth [61]. To counter these risks, Sweeney defined k-Anonymity as a property satisfied by a dataset `D` if, and only if, each sequence of values for quasi-identifiers in `D` is indistinguishable from at least $k-1$ other records in `D`. In practical terms, any tuple in `D` must be identical to at least $k-1$ other tuples concerning its quasi-identifiers. This property ensures that the data cannot be easily linked to other sources and protects users from re-identification, with the privacy guarantees strengthening as k increases.

Ahn, Bortz, and Hopper extended the concept of k-Anonymity to communication systems, defining a protocol as sender k-anonymous if it ensures that an adversary attempting to identify the sender of a message can narrow their search to a set of k potential senders. Similarly, receiver k-Anonymity guarantees that an adversary can only narrow down the possible recipients to a group of k [1].

Despite its foundational role, k-Anonymity remains vulnerable to several types of attacks, including unsorted matching, complementary release, and temporal attacks, as noted by Sweeney [60]. Unsorted matching attacks exploit the order of tuples in a released dataset, which can be mitigated by randomizing the tuple order. Complementary release attacks arise when subsequent data releases are combined, allowing adversaries to re-identify individuals by correlating datasets. Temporal attacks exploit data changes over time, such as additions, deletions, or modifications, leading to violations of k-Anonymity guarantees as datasets evolve.

Further research has explored enhancing k-Anonymity to address its limitations. For instance, Hopper and Vasserman observed that receiver k-Anonymity improves effectiveness against long-term intersection and statistical disclosure attacks. Additionally, incorporating periodic sender k-Anonymity under low churn conditions enhances resistance to mass surveillance, albeit without achieving an optimal cost function [24].

Recognizing that k-Anonymity alone may not suffice to protect privacy in the era of big data, Gosain and Chugh proposed combining k-Anonymity with Differential Privacy. This hybrid approach addresses the challenges posed by the massive scale and interconnected nature of modern data systems [22]. Similarly, in response to the proliferation of social networks and the accompanying surge in data collection, Campan and Truta introduced an anonymization technique for social network data, masking it according to the k-Anonymity model [6].

While k-Anonymity offers practical privacy guarantees, it is primarily suited for systems prioritizing efficiency over robust anonymity protections. It remains less applicable in scenarios where strong, comprehensive guarantees are essential.

2.4 Differential Privacy

Differential Privacy [16, 18, 43], introduced by Dwork et al., is a formal notion of privacy and is a property of algorithms, rather than data like k-Anonymity. An algorithm or function satisfies Differential Privacy if for all neighboring datasets x and x', and all possible sets of outputs S:

$$\frac{\Pr[F(x) \in S]}{\Pr[F(x') \in S]} \leq e^{\varepsilon} \tag{2.1}$$

In this privacy mechanism definition, F is a randomized function, and its output will be similar, with or without the data of any specific individual. Moreover, F's randomness should be enough so that outputs do not reveal the input data. As a result, we can ensure that the privacy of any individual is preserved, by ensuring plausible deniability. This happens because the output of the function cannot be correlated to any specific input, whether the input is present or not. Furthermore, the mechanism can maintain certain level of accuracy, due to having a precise understanding of the noise generation process. Differential Privacy intends to reject no malicious adversary, instead it ensures that the adversary is not capable of inferring any specific information, due to the randomness of the output.

Another important requirement of differential private mechanisms is the ϵ parameter [43, 18, 17, 16]. This parameter is called *privacy parameter*, and it controls the 'amount of privacy' that the correspondent function provides. Small values of ϵ require F to provide very similar output when given similar inputs, originating in higher levels of privacy. On the other hand, higher values of ϵ allow outputs to diverge more, leading to lower levels

of privacy. In practice, ϵ should be less or equal to 1, and no greater than 10, to provide meaningful privacy guarantees.

Differential Privacy originated from the challenge of enabling the extraction of meaningful insights about an underlying population while rigorously safeguarding individual privacy. Therefore, the earlier definitions of this concept were focused on datasets and queries over them.

The Laplace Mechanism was proposed together with the definition of Differential Privacy by Dwork et al. [18]. The simplest way to achieve Differential Privacy is by adding noise to the output of a function. The challenge is to balance the injected noise in order to have enough to satisfy the definition of DP but not too much to make the output useless. The Laplace Mechanism is a simple mechanism which, for a function $f(x)$ that returns a real number, $F(x)$ satisfies ϵ-Differential Privacy:

$$F(x) = f(x) + \text{Lap}(\frac{S}{\epsilon}) \tag{2.2}$$

where s is the sensitivity of the function f, and $Lap(S)$ denotes sampling from the Laplace distribution with scale S and center 0.

The sensitivity of a function is a measure of how much the output of the function can change when the input changes. This property is important to determine how much noise should be added to the output of the function, in order to guarantee Differential Privacy.

2.4.1 Properties of Differential Privacy

In this section, we cover the three main properties of Differential Privacy: Sequential Composition, Parallel Composition and Post-Processing.

2.4.1.1 Sequential Composition

Sequential Compositions [43, 17] is a major property of Differential Privacy, which determines the total privacy cost of releasing multiple results of differential private mechanisms on the same input. Formally, this property states that if $F_1(x)$ satisfies ϵ-Differential Privacy and $F_2(x)$ satisfies ϵ'-Differential Privacy, then the composition of these two functions, $G(x) = (F_1(x), F_2(x))$, satisfies $(\epsilon + \epsilon')$-Differential Privacy. This property is important to algorithms that access data more than once. The bound on privacy cost given by this property is an upper bound, and the actual privacy cost can be lower than the sum of the individual privacy cost.

2.4.1.2 Parallel Composition

Parallel Composition [43, 17] can be considered as an alternative to Sequential Composition, as it determines the total privacy cost of releasing multiple results of differential private mechanisms. The idea consists on splitting a dataset into disjoint chunks and applying a differentially private mechanism to each chunk separately. Formally, if $F(x)$ satisfies

ϵ-Differential Privacy, and we split the dataset X into k disjoint chunks such that $X = X_1 \cup X_2 \cup \ldots \cup X_k$, then the mechanism that releases all the results $F(x_1), \ldots F(x_k)$ satisfies ϵ-Differential Privacy. Comparatively, to Sequential Composition, Parallel Composition gives a better upper bound. Since F is run k times, Sequential Composition states that this procedure satisfies $k\epsilon$-Differential Privacy.

An issue with Differential Privacy arises with small datasets. A large dataset is able to achieve a strong privacy guarantee with relatively weak noise, and allows the results to be useful. However, small datasets require stronger noise to achieve the same privacy guarantee. As we split the dataset into chunks, we must care for the utility of the results and the impact of the number of chunks that we split the dataset into. The more chunks we split the dataset into, the stronger the noise we must add to the results, and the less useful the results will be.

2.4.1.3 Post-Processing

The last property is Post-Processing [43, 17], and it states that is impossible to reverse the privacy protection provided by a Differential Privacy by post-processing the data in some way. Formally, if $F(x)$ satisfies ϵ-Differential Privacy, then for any function h, $h(F(X))$ satisfies ϵ-Differential Privacy. This means that no danger will arise from performing additional computations on an output of a differential private mechanism, as the privacy guarantee will be preserved (and not reversed). This property is essential to ensure that Differential Privacy is resistant against privacy attacks based on auxiliary information and that attacks effectiveness is only limited by the privacy parameter ϵ.

2.4.2 Local Differential Privacy

Differential Privacy was initially proposed in a central model, where sensitive data was collected in a single dataset and the data curator must be a trusted entity in order to correctly execute differential private mechanisms, even in scenarios where the analyst is malicious [43]. However, this assumption might not be very realistic. In practice, the data curator and the analyst might be the same entity, and the data curator might not be trusted, leading to no differential private mechanisms being in place. Therefore, *Local Differential Privacy* (LDP) raises as a solution in which data is made Differential Privacy before being collected by the data curator, being already used by big companies such as Google [19] and Apple [27].

Randomized Response is a mechanism for LDP, proposed by Warner [67] in 1965, intended to improve survey responses about sensitive issues, but was not originally design as a mechanism for Differential Privacy. Dwork and Roth presented a variant of this mechanism, in which the subject flips a coin to answer a 'yes' or 'no' question. If the coin is heads, the subject answers truthfully, otherwise, the subject flips the coin again and answers 'yes' if heads and 'no' if tails. The randomization in this algorithm comes from the two coin flips, creating uncertainty about the true answer, and therefore providing privacy.

2.4.3 Differential Private Communication

Some studies have been conducted on applying Differential Privacy to communication systems [65, 71, 53]. These studies focus on providing privacy guarantees to users and protect them against traffic analysis attacks, powered by machine learning techniques.

Zhang et al. [71] proposed a differential private mechanism for streaming data, by the use of a Chrome extension with a DP mechanism that proxies streams between the browser and the server. The extension intercepts the requests from the client, which sends requests on behalf of the client based on a differentially private mechanism, instead of instantly relaying them immediately. The authors found that differentially privacy lowered the machine learning techniques, used by the adversary, efficiency, even though the accuracy of the attack was less affected in case of the adversary training model with the same DP mechanism.

NetShaper [53] is a network side-channel mitigation system, proposed by Sabzi et al., based on traffic shaping, which provides quantifiable and tunable privacy guarantees, through the use of Differential Privacy. This work shapes traffic based on a DP-mechanism that adds noise to the traffic, that can be split into 3 steps: *queue, query, and post-process*. In the first step, the system queues the input traffic. After a fixed parameterizable periodic interval, the system queries the queue and adds noise to the traffic, performed by the differential private traffic shaping algorithm. Finally, the packets are post-processed and transmitted to the network.

Vilalonga et al. [65] suggested Randomized Response could be used to strengthen the privacy of the Tor network, by adding noise to the traffic, making it harder for adversaries to correlate the traffic and de-anonymize users. The authors proposed a mechanism that uses pluggable transports to connect the client to the Tor network and a noise-adding algorithm. This algorithm sends packets based on the Randomized Response mechanism, in this case, where the algorithm chooses to send a packet or a noise packet based on a probability.

2.5 Summary

Anonymity networks have been around for almost 40 years but advances in machine learning and traffic analysis techniques have compromised the privacy guarantees that these networks aim to provide. Tor is a widely used low-latency anonymity network that has been shown to be vulnerable to such attacks. We take a particular interest in the developments on Tor's scheduling algorithms that proposed (and later been merged into the source code) solutions to improve Tor's congestion control. The works referred in this chapter have both demonstrated effects in performance and congestion control but lack on providing privacy guarantees that Tor aims to provide. Some work on anonymity network, including Tor, have tried to mitigate these attacks, such as website fingerprinting and correlation-based attacks, but they lack on giving formal privacy

guarantees. In most of the cases, privacy and security are only measured by practical testing, which may not be enough to ensure that the network is secure and private, if some conditions are not rigorously tested and proven. To address this problem, this dissertation introduces formal privacy guarantees to anonymity networks, more precisely to Tor, by implementing a Differential Private Tor Socket Scheduler that, which also offers adaptable and parameterizable privacy protections. To this day, there is no work that focus on applying Differential Privacy techniques into Tor network.

3

System Model & Software Architecture

This chapter provides an in-depth analysis of the system model and software architecture of TTT. We begin by introducing the system's overall objectives and foundational principles. Next, we define the design goals that guided its development, followed by a detailed breakdown of the system model and software architecture. Subsequently, we present the threat model, analyzing the potential threats and adversaries accounted for in TTT's design. Finally, we conclude with a comprehensive summary that synthesizes the key insights discussed throughout the chapter.

3.1 TTT Proposal

As discussed previously in section 1.1, Tor is a widely used anonymity network that provides users privacy and anonymity while browsing the internet. However, advances in machine learning and data analysis have made it possible to de-anonymize Tor users by analyzing their traffic patterns, therefore potentially compromising the privacy and anonymity that Tor aims to provide [REFS AQUI].

This can be achieved through some traffic analysis attacks, such as fingerprinting and traffic correlation. As discussed in section 1.2, these traffic analysis attacks represent a significant threat to Tor users' privacy and anonymity, as sophisticated adversaries can use the above-mentioned techniques to analyze monitored traffic patterns and de-anonymize users and their browsing activities.

Therefore, we propose TTT, an extension of Tor source code that includes a Differential Privacy TLS packet padding mechanism based on false Tor cells generation and two jitter injection Tor schedulers with various mathematical distributions to protect the users' data and prevent de-anonymization attacks. By incorporating Differential Privacy and injecting variable and randomized jitter into the Tor traffic, we have obfuscated its patterns and increased the network's resistance against fingerprinting and correlation attacks.

The main goal is to reinforce Tor's resistance against traffic analysis attacks, especially

fingerprinting and correlation, therefore fortifying users' privacy and anonymity, and ensuring that the solution maintains reasonable performance and usability. To achieve this goal, we designed TTT with the following goals in mind:

Formally Proven: The solution must be formally proven to provide a certain level of privacy, by using Differential Privacy. This way, we add a new and important layer of security and trust to the Tor network and users.

Tor's Extension: The solution must be an extension of the Tor project, respecting Tor's design principles and rules. As a very popular open source project, Tor has a large community of users and developers, and it is important to ensure that the solution can be easily integrated into the existing Tor source code.

Compatibility: The Tor network is composed of numerous voluntary relays operated by independent hosts. Therefore, the proposed solution must be easily integrated into the existing Tor infrastructure and remain compatible with relays that are unaware of it.

Unobservability: The solution must enhance Tor's resistance against traffic analysis attacks, specifically fingerprinting and correlation attacks, in comparison to the existing Tor network.

Configurability: As an extension of the Tor project, the solution must use the existing Tor configuration files and allow users and hosts to configure the trade-offs between privacy and performance, where users can choose the level of privacy they want to achieve.

Performance: We consider that performance is a key aspect of the solution. Even though the solution does not aim to improve the performance of the Tor network, to allow this solution to be relevant and useful for the Tor software and community, it is important to ensure that the solution does not significantly impact the performance of the Tor network.

3.2 System Model

As stated previously, the solution is an extension of the Tor project, so the system model is also similar to the Tor's system model, as briefly explained in Section 2.2. In order to preserve the Tor network's design principles and rules, our system model does not introduce any new components or mechanisms in the network layer of the system, as illustrated in Figure 3.1.

Instead, it focuses on enhancing the scheduling on existing Tor relays, with the proposed Differential Privacy Tor Cell Scheduler, and on introducing a new mechanism for generating additional traffic, called *Packet Padding Cells* mechanism, which are both presented further below.

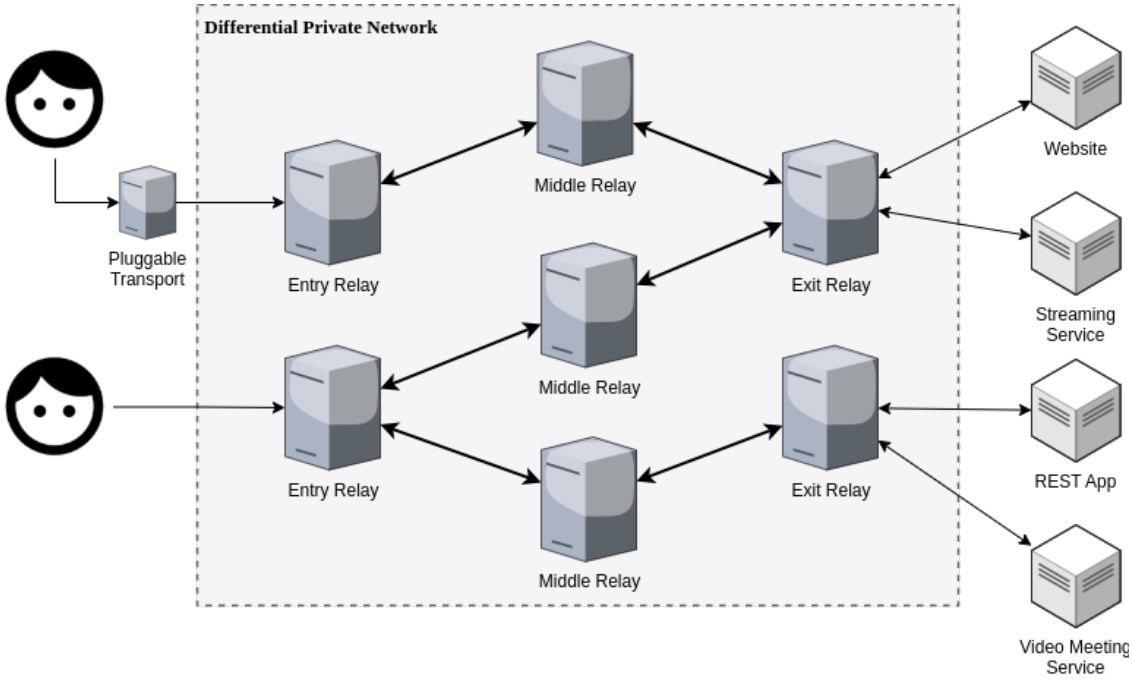


Figure 3.1: Differential Private Network System Model

Jitter Injection Scheduler: A pair of schedulers for Tor cells that incorporates mathematical distributions techniques to apply random jitter conditions to the Tor network traffic. By setting a random interval between scheduler runs, we are able to apply jitter to the Tor network traffic, making it more difficult for adversaries to analyze the traffic patterns and de-anonymize users. As each scheduler can or not apply a delay, the jitter is applied by each individual relay and/or client, making the privacy properties accumulate across the network, therefore providing a stronger privacy guarantee further the cell lives and travels through the network.

Packet Padding Cells (PPC): A mechanism that generates additional traffic by creating false Tor cells to the network traffic. These cells are generated based on a decision-making differential private mechanism triggered when the relay receives a new Tor `relay` cell. The newly generated cells are added to the circuit cell queue of the original cell, which leads to the generated traffic to only be added to active and established circuits. In comparison to the above schedulers, the privacy properties do not accumulate across the network, as the cells are generated only when a new Tor cell is received, and discarded on reception. In this case, 2 adjacent segments will have the same amount of Packet Padding Cells, but the ones created by the initial relay will be discarded by the second and the last relay will only receive the false cells generated by the second relay, also meaning that the TLS packet count won't be coupled to the number of cells created nor to a certain pattern shared by all segments. Additionally, the client does not produce any additional traffic, unlike the scheduler.

These two new features are designed to work independently, meaning that the user can

choose to use one or both of them, but with the same goal of enhancing the Tor network's resistance against traffic analysis attacks.

3.3 Threat Model

As mentioned earlier, Tor network is widely used anonymity network that may be vulnerable to traffic analysis attacks, such as fingerprinting and traffic correlation. Considering these types of attacks, adversaries analyze and monitor traffic in certain segments of the network, aiming to gather information to train and strengthen their machine and deep learning models, which can then be used to de-anonymize users and expose their browsing activities, for example. This way, users that browse the internet through the Tor network may be vulnerable to these attacks, which can lead to the exposure of their browsing activities and identities, and making Tor inefficient.

TTT is designed to strengthen the Tor network's privacy guarantees and bolster its defenses against traffic analysis attacks, with a particular focus on the before mentioned threats. While developing our threat model, we closely follow the foundational Tor threat model outlined by [15] [15], taking into consideration adversaries whose ultimate objective is the de-anonymization of users and the exposure of their browsing activities, and which are able to monitor portions of network traffic, manipulate data flows, operate their own onion routers, and compromise a fraction of existing routers.

Before defining our adversary model, we first clarify the types of adversaries considered and which are not included in our threat model. State-Level adversaries (SLAd) are capable of observing Tor's traffic patterns in one or few Autonomous Systems (AS). Large-Scope adversaries (LSAd) are considered a group of SLAd that can monitor a significant fraction of the network, by collaborating and working together and therefore controlling the collection of regions controlled by each collaborator. Finally, Omnipresent adversaries (OPAd) are a powerful attacker, or a group of attackers, that can monitor the entire Tor network, including all segments and regions, and therefore can observe all traffic patterns.

Figure 3.2 illustrates the threat model over the system model, showing the adversaries' capabilities and the Tor network segments that they may monitor. The red magnifying glasses indicate the more commonly observed segments of the network, whilst the orange magnifying glasses represent the segments that are less commonly observed. Nonetheless, it is important to note that the adversaries considered by our threat model are not limited to the firstly described segments and our adversaries can monitor any pair of segments in the network.

For the scope of our solution, we considered SLAd to be the main adversary, as they are more common and realistic in the context of Tor. LSAd are also considered to be a threat and also included in the scope of our threat model, even though we recognize that they are less common. On the other hand, OPAd are not considered in our threat model, as we consider they are not realistic and do not represent a common threat to Tor users.

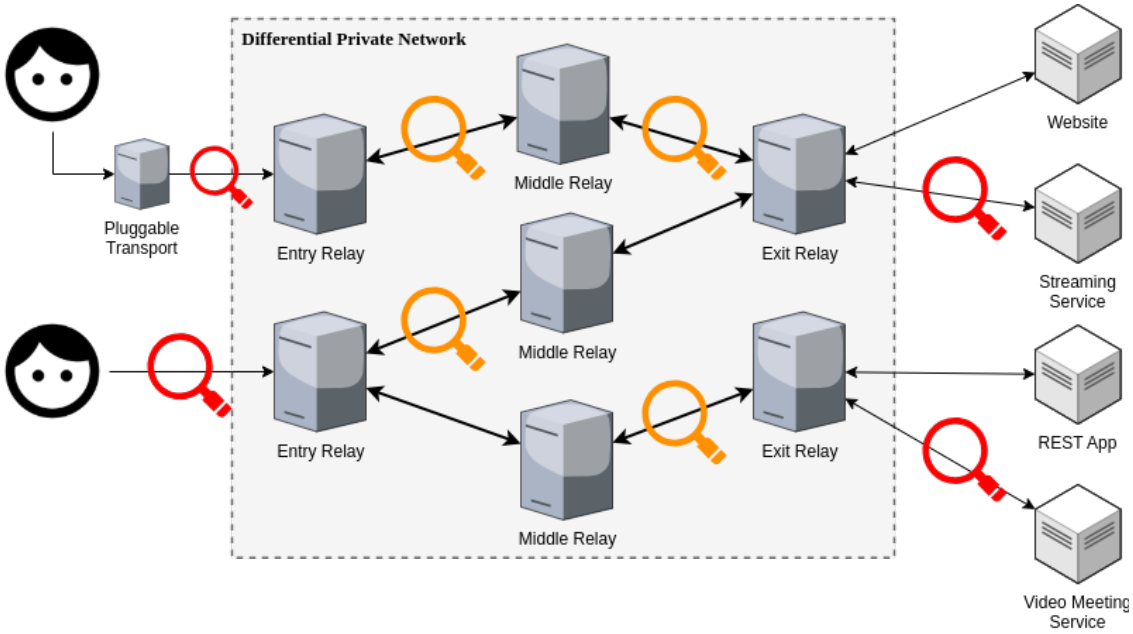


Figure 3.2: Threat Model over System Model

Building on the original assumptions put forth by Dingledine, Mathewson, and Syverson [15], we consider both passive and active adversaries. A passive adversary merely observes traffic patterns, applying machine learning and data analysis techniques to infer user identities and behaviors, without being able to observe the content of TLS packets. On the other hand, an active adversary compromises onion routers to inject traffic into the network — not merely to observe, but to facilitate more sophisticated traffic analysis attacks. Traffic injection, in this context, becomes a strategic tool for the adversary to train machine learning models capable of uncovering users' identities and tracking their navigation through the Tor network. However, it is important to note that we do not take relay compromise into account. In some context, adversaries may deploy a malicious voluntary relay and our solution is not intended to defend users, as the attacker would be able to monitor the traffic patterns and defeat the Packet Padding Cells mechanism.

In summary, our threat model considers powerful State-Level and Large-Scope adversaries capable of monitoring significant portions of the Tor network traffic, but not on its entirety. These adversaries are equipped with advanced and state-of-the-art machine and deep learning and data analysis techniques. These adversaries may be passive, observing traffic patterns, or active by injecting traffic into the network to train their machine learning models.

3.4 Software Architecture

To ensure a clear understanding of the software architecture of TTT, we will first present the Tor relay architecture, which is the foundation of TTT, and then we will present the TTT architecture, as an extension of the Tor relay architecture.

3.4.1 Tor Relay Architecture

Tor relays are the backbone of the Tor network, responsible for routing traffic between clients and servers while maintaining user anonymity. As explained in greater detail in section 2.2, Tor relays are connected through TCP over TLS connections, forming a network of relays that route TLS packets composed by a variable number of cells between clients, relays and servers, encrypting and decrypting the traffic at each hop, layer by layer.

Our solution extends the Tor relay architecture by introducing a pair of new Tor schedulers and a new mechanism for generating additional traffic, presented in section 3.2. At the time of writing, Tor has 2 types of schedulers: the *KIST* scheduler, briefly presented in subsubsection 2.2.3.2, which also has a *Lite-KIST* variant for low-end devices, and the *Vanilla* scheduler, which is a simpler scheduler. To develop our solution, we chose to extend both scheduler to provide better insights about the schedulers' performance with our solution, and to allow users to choose the scheduler that best fits their needs.

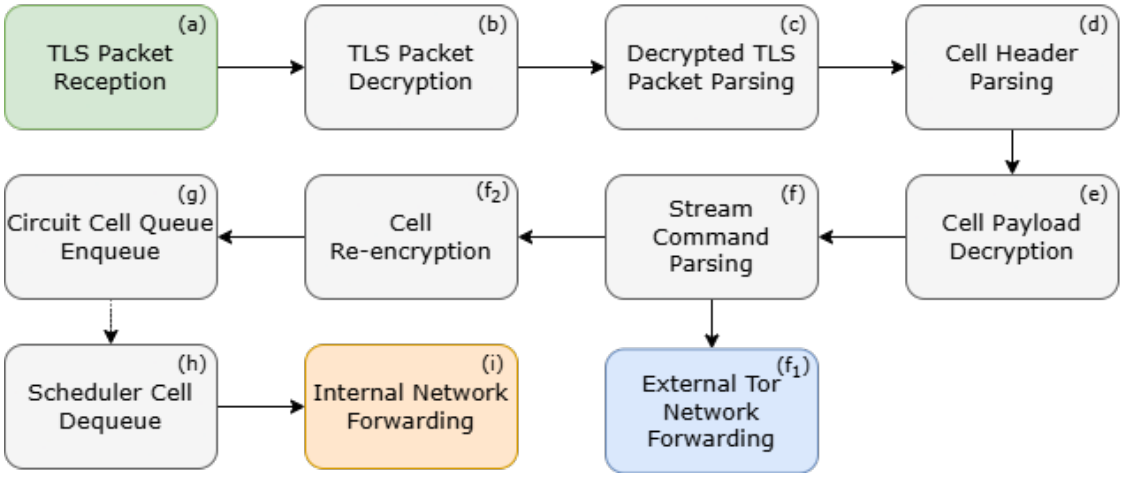


Figure 3.3: Original Cell Processing Pipeline

To ease the understanding of the Tor relay architecture, we present an illustration of the pipeline of a Tor cell in a Tor relay in Figure 3.3. As shown in the figure, after receiving a TLS packet (a), it gets decrypted (b) and all Tor cells within the packet are extracted (c). For each Tor cell within the packet, its header is checked (d) and its payload gets decrypted (e). After decrypting the payload, the cell's header *stream command* is used to determine if the relay is the exit and the cell must be forwards to the destination (f_1), or if the relay corresponds to an entry or middle relay, in which case the cell is forwarded to the next hop. Finally, the cells that must be forwarded are re-encrypted (f_2) and enqueued in the respective circuit cell queue (g). The scheduler then dequeues the cells from the circuit cell queue (h), and forwards them to the next hop (i).

3.4.2 TTT Architecture

After presenting the Tor relay architecture, together with a better understanding of the Tor cell lifetime in a relay, we can now present the TTT architecture. As stated in section 3.2, our solution brings 2 main features to the Tor project: a new scheduler that applies jitter to the Tor network traffic, and a new mechanism for generation additional traffic.

To allow a better comprehension and comparison to the differences of Tor cells lifetime in the original Tor project and in TTT, we present a similar illustration to Figure 3.3 in Figure 3.4, which illustrates the pipeline of a Tor cell in a TTT relay and identifies the differences introduced by our solution.

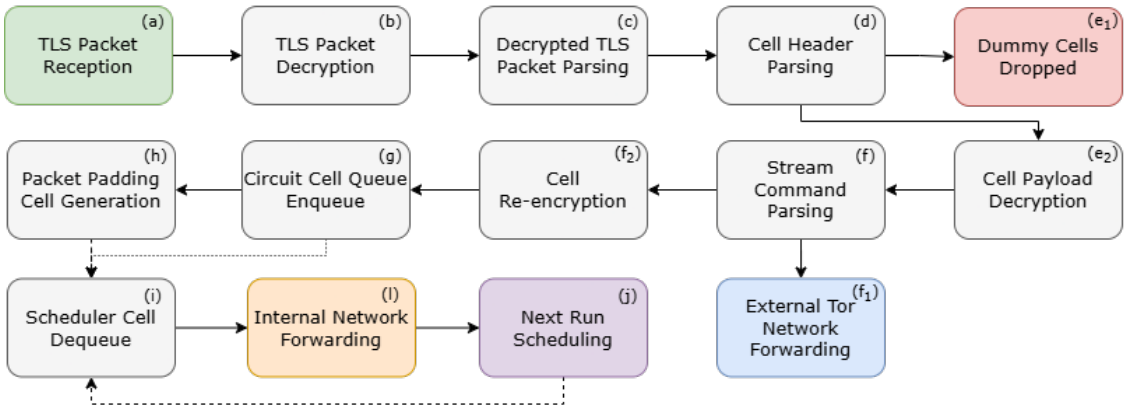


Figure 3.4: TTT Cell Processing Pipeline

Upon receiving a TLS packet (a), the relay decrypts it (b) and extracts all cells within the packet (c), same as the original Tor. For each cell within the packet, the relay checks its header (d). If the cell is a false cell, it is discarded (e_1), otherwise the cell is decrypted (e_2). After decrypting the cell, its *stream command* header is checked to determine if the relay is the exit and the cell must be forwarded to the destination (f_1), or if the relay corresponds to an entry or middle relay, in which case the cell is forwarded to the next hop. For the cells that must be forwarded, they are re-encrypted (f_2) and enqueued in the respective circuit cell queue (g). After queuing a cell, the relay decides, based on a differential private algorithm, if it generates a new false cell (h). Newly generated cells are added to the circuit cell queue of the original cell. Once any cell is added to the circuit cell queue, the circuit is marked to inform other components that it is ready to sent cells outwards, leading the scheduler to dequeue the cells from the circuit cell queue (i). After being dequeued, cells are placed in the output buffer for retransmission, and forwarded the cells to the next hop (l). Meanwhile, the scheduler calculates if jitter must be applied to the circuit, therefore setting a random interval between the time of the last scheduler run and the next, based on a configurable and pre-determined mathematical distribution and randomness parameters (j), impacting the following cells time of retransmission.

In comparison to the original Tor cell pipeline, the main differences are the step (e_1), where the false cells are discarded, the step (i), where the process of decision and

generation of *Packet Padding Cells* takes place, and the step (m), where the scheduler randomly decides when the next run will be. The scheduler implementation of jitter is originated by scheduling the next time the scheduler will run again based on a random delay, which is calculated based on the differential private algorithm. This way, the scheduler can apply jitter to the Tor network traffic and be non-blocking to other relay processes essential for the Tor software to work properly.

3.5 Parameterization

Our solution extends Tor with 2 new features to enhance its resistance against traffic analysis attacks, and we aim to provide users and hosts with full control over the privacy and performance trade-offs. To do so, the parameterization of our solution must be user-friendly and easy to configure, without requesting users to have a deep understanding of the underlying mechanisms

3.5.1 Used Techniques

The Tor project handles its configuration through a key-value file called `torrc`. This file is used as a single point of configuration for the Tor network, allowing users and hosts to define the type of relay, to set various parameters such as bandwidth limits and to configure the behavior of the Tor software. Some configuration parameters also allow the software to swap configuration mid-execution, without the need to restart the Tor process.

Our solution extended the existing Tor configuration file to include new parameters that allow users and hosts to configure the behavior of the new schedulers and the Packet Padding Cells mechanism. These parameters are designed to give full control to users and hosts, allowing them to choose the level of privacy they want to achieve.

3.5.2 Jitter Parameterization

To configure the Tor software to use our schedulers and configure the injection of jitter into the network traffic, we extended the torrc file with the following parameters:

Schedulers As mentioned before, this parameter was already present in the Tor project, and was extended to accept the new schedulers, `PRIV_KIST` and `PRIV_Vanilla`.

PrivSchedulerEpsilon Similar to the `DummyCellEpsilon` parameter, this parameter is used to define the variability of the jitter applied by the scheduler. As an inspired Differential Privacy parameter, the value must be greater than 0, and the higher the value, the less jitter will be applied, therefore the less privacy will be provided. The default value is set to -1, which turns the jitter off. When the jitter is turned off, the scheduler will behave as its original version.

PrivSchedulerDistribution This parameter is used to define the distribution used to generate the time intervals between scheduler runs. At the moment, the schedulers only accept the following distributions: *Uniform*, *Laplace*, *Poisson*, *Exponential* and *Normal*. By default, if one of our schedulers is used and no distribution is set, the Tor software will exit with an error.

PrivSchedulerMinJitter This parameter sets the minimum possible interval between two consecutive scheduler runs. The default value is set to 1.

PrivSchedulerMaxJitter Similarly to the `PrivSchedulerMinJitter` parameter, this parameter sets the maximum possible interval between two consecutive scheduler runs. The default value is set to 10.

PrivSchedulerTargetJitter To give more control to the user, this parameter is used to set the target jitter that the scheduler should aim to achieve. The scheduler will try to apply a random delay between the minimum and maximum jitter, with center in this parameter. The default value is set to 2.

These last 3 parameters are tightly related and the Tor software will exit with an error if `PrivSchedulerMinJitter` $\leq$ `PrivSchedulerTargetJitter` $\leq$ `PrivSchedulerMaxJitter`.

These parameters allow users to configure the jitter generation process, by defining a randomness Differential Privacy-inspired variable and an interval of possible outcomes. We also integrated some of the most common mathematical distributions to generate the random numbers, allowing users to choose the distribution that best fits their needs and to allow the jitter conditions among several network segments to behave differently.

3.5.3 Packet Padding Cells Parameterization

The *Packet Padding Cells* feature is a simple mechanism that generates additional traffic with a Differential Private Randomized Response mechanism. To keep this feature as simple and user-friendly as possible, we extended the torrc file with the following parameter:

DummyCellEpsilon This is the only configuration parameter used for the Packet Padding Cells mechanism. It is used to set the Differential Privacy parameter ϵ value for the decision-making algorithm that generates the Packet Padding Cells. By default, the value is set to -1, which turns the mechanism off. As a Differential Privacy parameter, the value must be greater than 0, and the higher the value, the fewer cells will be generated, therefore the fewer privacy will be provided. The highest possible ratio of generated cells in relation to the original cells is 0.5, when this parameter is set to 0.

With just one parameter, users and host are able to configure the Packet Padding Cells mechanism, allowing them to choose the level of privacy they want to achieve.

3.6 Jitter Development Strategy

3.7 Packet Padding Cells Development Strategy

3.8 Discussion

4

TTT Prototype and Implementation

In this chapter, we explore the implementation and prototyping details of TTT solution. We provide a comprehensive overview of the design choices, technologies used, and the development process that led to the creation of the TTT prototype. We also discuss the complexities involved in the implementation and how we addressed them to ensure a robust and effective solution. Finally, address the availability of the prototype for further research and development with an in depth guide on the installation and setup process, as well as a walkthrough on the deployment of a private network for validation and testing of this work.

4.1 Prototype Overview

The TTT prototype is designed to enhance the Tor network's against traffic analysis attacks by implementing 2 features on the Tor software: the **TTT Privacy Schedulers** and the **Packet Padding Cells** (PPCs). We extended the Tor source code, version `0.4.9.1-alpha-dev`, using `C` programming language, and integrated these components, together with a simple and modular library and a set of helping scripts for testing, preserving the original Tor's stack, extending its functionalities and improving privacy guarantees.

Tor Schedulers are responsible for deciding when circuits should send pending traffic and handling the circuits' cell queues. Our TTT schedulers are very similar to the original and KIST Tor schedulers, but, instead of running on fixed time intervals, these intervals between executions are variable and randomized. All Tor's schedulers consist on a collection of function pointers, which are used to call the appropriate functions for each scheduler. Among these functions, every implementation must provide two mandatory functions: `schedule` and `run`. As the name suggests, the `schedule` function is responsible for scheduling the next execution of the scheduler, while the `run` function is the most important, and it is responsible for executing the scheduler logic, which includes deciding which circuits should send pending traffic. In our solution, we focus on surgically modifying the `scheduler` function to calculate and schedule the next execution based on a pseudo-random time interval, based on the user's configuration. Also, Tor unpacks TLS

packets, retrieves all cells that it may contain, and then assigns each cell to a correspondent circuit queue. Our Packet Padding Cells prototype modifies this pipeline by adding an extra step, where a false copy of the cells is created and added the queue.

To respect the modular design of Tor and its community guidelines, we also implemented a library of pseudo-random number generation based on mathematical distributions and Differential Privacy mechanisms. This library contains 2 main public functions: `dp_generate_int` and `dp_generate_bool`, which generate a pseudo-random integer and boolean, respectively. The first one uses a given minimum, maximum and target integers, an ϵ value and the mathematical distribution to generate a pseudo-random integer, inside the given interval, with center in target and uses ϵ as a Differential Privacy-inspired privacy parameter. The second one only uses a given ϵ value and a true decision to generate a differential private boolean. These functions are used in the TTT Schedulers and PPCs to generate randomized decisions, respectively, but may be used in other components in the future, given the modular design of the library. The Tor Configuration, as mentioned in the previous chapter, uses a `torrc` file to configure the Tor software. We extended this configuration file with the before-mentioned parameters.

Finally, to assist the implementation and testing of our solution in a more efficient and automated way, we also developed a set of scripts for validation, testing, extracting and plotting the results of the TTT prototype. This collection of scripts can be split into 3 individual scripts. The *Tester* is a *Bash* script that runs the TTT prototype in a *Docker Swarm* or *Docker Compose* environment, using the implemented Tor code. This script uses a *YAML* configuration file in order to allow the user to specify the parameters of the TTT prototype, such as the number of `curl` requests, the size of the file to download, all needed PPC and Scheduler parameters, for each test, among other parameters. It supports both individual configuration of each test and a parameter combination mode, where the user can specify a set of parameters and the script will generate all possible combinations of those parameters to run the tests. This complex script is also responsible for building the needed *Docker* images and for collecting the output of the tests and storing them in a predefined folder. The *Analyzer* is a *Python* script that parses the output of the previous script, extracting the relevant performance information from the `curl` output, such as the *HTTP* code, the throughput, the total time and the time to first byte of each request. The Tor relays' logs of each test are also parsed, extracting the relevant information such as the number of TLS packets and cells received, the number of Packet Padding Cells created, and the jitter condition experienced of each node. Finally, this script generates a *JSON* file with all the extracted information, such as mean, minimum, maximum and standard deviation of each metric, for each test, which is then used by the *Plotter* script to generate the plots. The *Plotter* is also a *Python* script that uses the *Pandas* and *Matplotlib* libraries to generate plots, such as the ones used in chapter 5.

4.2 Technologies and Techniques

The TTT prototype is an extension of the Tor project, developed in *C* programming language, leveraging the existing Tor codebase and libraries, and adding new features and functionalities.

To implement both the TTT KIST and Vanilla schedulers, we created two new files: `scheduler_privacy_kist.c` and `scheduler_privacy_vanilla.c`, each inspired by the original KIST and Vanilla schedulers, respectively, modifying some functions such as the ones stated in the previous section. To enhance the privacy the original schedulers provided, we used a mathematical distribution to generate a pseudo-random time interval, in the `schedule` function, for the next execution of the scheduler. After each scheduler execution, the scheduler is responsible for scheduling its next execution, and with our modification, we are able to apply a variable, random and different time interval between each scheduling. Over time, this allows for a more dynamic and less predictable scheduling pattern, which are reproduced in the traffic that exits the node. It is also important to note that the time intervals is only set after the scheduling of a scheduler run, to avoid performance time penalties.

By extending the `relay.c` file, which handles cell encryption and decryption, packaging and receiving from circuits and queuing on circuits, we were able to implement the Packet Padding Cells feature. We modified the `circuit_receive_relay_cell` function by adding an extra step after successfully queuing a received cell, where accordingly with a given probability, a new cell with a random payload is created and enqueued in the same circuit queue of the original cell. This probability is defined by the `DummyCellEpsilon` configuration parameter which then is used to generate a differential private boolean, using the `dp_generate_bool` function, to decide whether to create a new cell or not.

Our library gives Tor source code the possibility to generate pseudo-random decisions and numbers, using Differential Privacy and mathematical distributions. At the moment of writing, the library supports the *Laplace*, *Poisson*, *Exponential*, *Uniform* and *Normal* distributions for generating pseudo-random numbers, and the Randomized Response algorithm for generating pseudo-random decisions, as showed in Listing 4.1.

```
1  bool randomized_response(bool true_value, double epsilon)
2  {
3    if (epsilon < 0.0) {
4      return true_value; // If epsilon is negative, return the true value
5    }
6    double p = exp(epsilon) / (exp(epsilon) + 1.0);
7
8    if (uniform() < p) {
9      return true_value;
10   }
11   return !true_value;
12 }
```

Listing 4.1: Randomized Response algorithm implementation.

To achieve a configurable and flexible solution, we extended the Tor configuration file, `torrc`, with new parameters that allow users to configure the Tor program with the TTT prototype features. Thus, we extended the `config.c` and `or_options_st.h` which are responsible for parsing the Tor configuration file and storing the options in the `or_options` structure during runtime, respectively, to accept new configuration keys listed in section 3.5.

The Tester script is written in Bash and uses *Docker Swarm* or *Docker Compose* to deploy the Tor network with the TTT prototype. It uses the `yq` and `jq` tools to parse the *YAML* and *JSON* files, respectively, and the `curl` command to execute the tests' *HTTP* requests and to simulate user traffic, such as browsing or download a file. It also uses the `grep` command to parse the state of the network based on the Tor logs, `ssh` and `scp` commands to interact and transfer files from the different rented machines used for testing. By using *Docker*, we are able to create a controlled and easy-to-deploy environment, both for an emulated and local and a distributed deployment, allowing us to run the tests in a consistent and reproducible way for all test bench scenarios. The *Analyzer* scripts is written in *Python* and uses *NumPy* library to calculate the statistics of the extracted data such as mean and standard deviation (for throughput and total time) and 10th, 50th and 90th percentiles (for time to first byte). The *Plotter* script is written in *Python* and uses *Pandas* and *Matplotlib* libraries to generate plots based on the *Analyzer* script output.

The deployment was performed using *Docker* given its flexibility and ease of use, allowing us to create a controlled environment for testing and validation, for both emulated and distributed scenarios. We also considered using Shadow [30] given that it was the tool used to test the KIST scheduler [32] but ultimately decided against it due to its complexity and lack of expertise in comparison to the *Docker* ecosystem. Furthermore, using Docker lead to a very smooth and easy transition between both scenarios, but only changing Docker Compose files.

4.3 Complexity Analysis

To implement, validate and test the TTT prototype, we extended the Tor source code with 2 new schedulers and the Packet Padding Cells feature, extended the Tor configuration file and created a library of mathematical distributions and Differential Privacy mechanisms. We also developed a set of scripts to automate the testing, validation and analysis of the TTT prototype. To understand the complexity of our work we break down the more technical details of the presented work.

To successfully extend the Tor source coded and implement the abovementioned features, we had to understand the Tor architecture, how each component works, interact and communicate with each other, and how to integrate our changes into the existing codebase. The Tor source code is well documented, but it is still a complex codebase, with many components and interactions. The `0.4.9.1-alpha-dev` branch of the Tor source code contains nearly 1 600 files, from which 645 are *C* source code files and 569 are

header files, and a near total of 360 000 lines of code, including comments and empty lines, from which 290 000 correspond to *C* source code files and 31 820 to *C* header files. Besides *C* source and header files, the Tor source code also contains *Python* scripts, *YAML* configuration files, and other auxiliary files. Even though the Tor source code is a very large and vast codebase, its modular and well documented structure minimizes its complexity and allows easier and faster understanding and extension. For example, inside the `src/lib` folder, where all libraries are stored, three is 448 files and 44 616 lines of code, while the entire `src` folder, which contains all the source code, has 1 458 files and 335 284 lines of code.

Our version of the source code, with the TTT prototype, added two files in the `src/or/core` folder: `scheduler_privacy_kist.c`, `scheduler_privacy_vanilla.c`; and `dp_mech.c` and `dp_mech.h` in the `src/lib` folder. The first 2 files contain the implementation of the TTT schedulers, with a total of 584 lines of code. The last 2 files contain the implementation of our random number generation library with a total 236 lines of code.

In addition to the added files, the `src/or/core/relay.c` file was modified to implement the Packet Padding Cells feature, with an average of 121 added lines of code and 475 modified lines of code. We also modified the `src/or/config.c` and `src/or/or_options_st.h` files to extend the Tor configuration, with nearly 90 lines of code added or modified.

Regarding the developed scripts, the *Tester* script has around 640 lines of code out of 7 *Bash* files, the *Analyzer* script has 460 lines of code among 6 *Python* files, and the *Plotter* script has 416 lines of code out of 3 *Python* files.

In total, this dissertation's work have developed 4 new *C* files and 16 *Bash* files, modified 3 *C* files, leading to the addition and modification of more than 1 400 lines of code in the Tor source code and more than 1 500 lines of code in the scripts. We also developed all the needed *Dockerfiles* and configuration files to run the TTT prototype in a *Docker Swarm* or *Docker Compose* environment, allowing for an easy and reproducible deployment of the Tor network with the TTT prototype.

In summary, the implementation of the TTT prototype required a deep understanding of the Tor architecture and mainly the scheduler component and the cell pipeline, as well as the architecture goals and rules of the Tor project. The development and validation of this implementation also required the study and understanding of the Differential Privacy mechanisms and mathematical distributions, as well as the reading of the Docker documentation to understand how to deploy the Tor network with the TTT prototype. This complexity study was performed using the `git diff` command to compare the actual source code with the last commit preceding the implementation of the TTT prototype, and the `cloc` command [14] to count the lines of code in the source code and scripts.

4.4 Prototype Availability

The prototype was made available to the public to allow for further research and development and to demonstrate how to modify Tor source code and how to deploy an

experimental and private Tor network with any version of source code. All extend of the present work is divided in two repositories: a *GitHub* repository with all the configuration files, scripts, the report and the Dockerfiles, and a *GitLab* repository with the code of the prototype. The Tor Project official repository is available in GitLab, therefore our fork is also available there, allowing for an easy integration with the Tor Project's development process and community.

To guide anyone interested in using the TTT prototype, or reproducing the results of this dissertation, in the following sections, we explain the installation and setup process, as well as the deployment of the Tor network with the TTT prototype.

4.4.1 Installation and Setup

Firstly, to create a private network, you must have a set of configuration files ready to be copied and used by the Tor nodes. We already provided a set of configuration files in the `testing/configuration` folder of the *GitHub* repository, which we strongly advise you to clone and use as a starting point.

```
$ git clone https://github.com/GoncaloRodri/Thesis.git
```

After cloning the repository, you may also clone into the repository the one we provided in the *GitLab* repository[1].

```
$ git clone <tor_repository> tor
```

Before running the Tor network, you must install the needed dependencies. One of the biggest dependencies is `Docker`, which we recommend you to install using the official documentation[2]. To run the *Tester* script, you will need to run the following:

```
$ apt install jq curl ssh openssl-devel-engine
$ snap install yq # package yq from apt may have compactability issues
```

By using *Docker*, we do not need to install all Tor's dependencies, as they are already included in the *Docker* image. After installing all the needed dependencies, now you can deploy the Tor network with the TTT prototype and perform any testing that you may consider adequate.

4.4.2 Deployment for Validation and Testing

If you want to deploy the presented work in a *Docker* environment, you may use the *Tester* script to deploy the network. In case of testing in a local scenario, with all nodes

[1] https://gitlab.torproject.org/GoncaloRodri/differential-privacy-tor
[2] https://docs.docker.com/engine/install/

containerized in the same machine, you may use *Docker Compose*, otherwise, you may use *Docker Swarm*. If using the latter, you must initiate the *swarm*, join all other nodes and create a configuration file or use one of the provided in the `testing/scripts/tester` folder. To build all necessary Docker images from the Dockerfiles in the repository and to create a swarm and join, you must run the following commands:

```
$ docker buildx build \
      -t <docker_username>/dptor_base \
      -f testing/docker/base.Dockerfile \
      .
$  docker buildx build \
      -t <docker_username>/dptor_node \
      -f testing/docker/node.Dockerfile \
      .
$  docker buildx build \
      -t <docker_username>/dptor_swarm \
      -f testing/docker/swarm.Dockerfile \
      .

$ docker swarm init
  ## In the other nodes
$ docker swarm join --token <token> <manager_ip>:<manager_port>
```

It is highly recommended setting a hostname for each machine to manually allocate each container to a determined physical machine. You can set the hostname by running the following command on the swarm manager:

```
$ docker node update --label-add <machine_hostname> <docker_node_id>
```

In case of local deployment, you just need to modify the tester script to use *Docker Compose*, the correct compose file and the only preparation needed is to ensure that the Docker daemon is running, and all Docker images were built.

Finally, to run the *Tester* script and deploy the network, you may run the following command:

```
$ ./testing/scripts/tester/monitor.sh -c <configuration_file>
```

However, you are also able to run the Tor network with the TTT prototype without using the *Tester* script, by using only the available configuration files and running the following set of commands, after changing the torrc files accordingly to your needs:

```
$ docker stack rm -d=false thesis
$ docker build -f docker/swarm.Dockerfile -t dptor_swarm --no-cache .
$ docker stack deploy \
    -c testing/scripts/tester/swarm.docker-compose.yml \
```

```
    thesis
# or
$ docker compose up
```

After running the above commands, your network will be deployed and start bootstrapping. You can check the status of the network by analyzing the tor logs, which contain the phase of bootstrapping of each node. A node is ready when it reaches the `'Bootstrapped 100%'` phase. We consider the network bootstrapped when all nodes are ready, which may take a few minutes, depending on the number of nodes and the network conditions. To check the status of the network, you can run the following command:

```
# performed for each relay
$ grep -i "Bootstrapped 100%" <path_to_relay_logs>
```

Finally, when the network is bootstrapped, you may use the network to run your tests. To test our prototype, we used the `curl` command to download a file from a web server, as the following command:

```
$ curl -o <output_file> --socks5 <socks_proxy> <url>
```

We recommend you to run the command in the Tor client machine. If so, as the client container shares the port 9000 to its host machine, this machine will use its container as proxy. This way, you must set the `socks_proxy` to `127.0.0.1:9000`.

Finally, it is important to note that Tor network may take long to bootstrap, but the configuration files can also influence this time. To accelerate the process, we expect all nodes to bootstrap in less than 5 minutes, by trying multiple times until successfully connecting all nodes under the mentioned conditions. We also recommend to firstly try to deploy the network locally, to validate the solution before deploying it in a distributed environment.

4.5 Summary

In this chapter, we explored the implementation and prototyping details of the TTT solution by providing a detailed overview of the technical choices, technologies used, and the development process that led to the creation of the TTT prototype. We discussed the complexities involved in the implementation, the size of the Tor source code, and how we addressed them to ensure a robust and effective solution. We also explained the installation and setup process for the prototype, the deployment for validation and testing, and the availability of the prototype for further research and development.

5

Validation and Experimental Evaluation

In this chapter, we present the validation and experimental evaluation of the proposed solution. We start by discussing the evaluation criteria used to assess the quality of the present work, such as the used metrics to perform such measurements and the test benches employed. Then, we present the performance observations and results according to such criteria. Finally, we discuss the unobservability evaluation, and the formal validation of the proposed solution.

5.1 Evaluation Criteria

To assess the quality of several aspects of the proposed solution, we defined a set of evaluation criteria as well as their importance to the overall system validation and evaluation. Tor is designed to protect users' privacy and anonymity, especially while web browsing, therefore is also important to evaluate the impact on performance. Tor already maintains a set of performance metrics and continuous evaluation, shared through Tor Metrics[1]. Among several metrics and data collected by this project, we selected the most relevant and simple ones to compare our solution. Regarding performance, we used the standardized activity to evaluate the performance: *download a file*. Tor Metrics uses 3 different file sizes to evaluate Tor: 50 KiB, 1 MiB and 5 MiB. Given this, we also tested our solution by downloading files of these sizes and compared the results regarding the throughput, the total time to download such files and the circuit round-trip latencies of circuits.

Also, given the solution's goal to enhance Tor users' privacy and anonymity guarantees, we also considered the impact on the unobservability of the traffic. To evaluate the resistance of the solution, especially against attacks targeting website fingerprinting, such as the ones pointed out throughout this work as those carried by our adversaries, we conducted fingerprinting resistance tests. This tests allowed for a better understanding of

[1] https://metrics.torproject.org/

the impact of the proposed solution on the unobservability of the traffic, and therefore on the anonymity guarantees.

Finally, a formal validation of the proposed system was performed to mathematically express its privacy guarantees, leveraging established theorems of Differential Privacy.

5.1.1 Test Benches

To conduct the before-mentioned tests, we design a couple of test benches to enlarge the scope of validation and evaluation of the solution. The testing environment for all test benches was designed to approximate a real-world scenario as closely as possible and to facilitate the easy replication and deployment of the experiments and their corresponding results. This was achieved by deploying two types of Tor networks:

Local Simulated Network This tests bench focused on validating the solutions' extension of Tor, by simulating a minimal Tor network, with 4 relays (directory authority, 2 non-exit relays and 1 exit relay) and Tor client on a Docker Composed system on a single machine. Although this scenario does not replicate the exact conditions of a live Tor network, it served as a crucial preliminary stage for establishing and validating the deployment process of a private, minimal Tor network and for verifying the fundamental functionality of our proposed extensions. This test bench was performed on a single 2 vCores OVH[2] Virtual Private Server (VPS) with 4 GB of RAM, configured with Ubuntu 24.10.

Distributed Network This test bench aimed to evaluate our solution in a more realistic scenario, by emulating a small private distributed Tor network, with 4 relays and a Tor client, each deployed on a separate OVH VPS using Docker Swarm. On the tests. This test bench allowed us to evaluate the performance and unobservability of our solution in a more realistic environment, with network latencies and conditions closer to those of the real Tor network. This test bench was performed using 5 OVH VPSs, each with 2 vCores and 4 GB of RAM, configured with Ubuntu 24.10. These machines were dispersed through France, Germany, Poland and the United Kingdom.

5.1.2 Experimental Observations

Our solution produced results for both performance and unobservability evaluations. The performance observations were performed using the `curl` command to download files of different sizes. The files were generated by a simple *Python* web server hosted by our private network as a container, present in the directory authority machine, in case of the distributed test bench. The used command was:

```
$  curl --socks5 <tor_proxy> -H 'Cache-Control: no-cache' \
      -w 'Code: %{response_code}\n
         Time to first byte: %{time_starttransfer}s\n
```

[2] https://www.ovhcloud.com/pt/

```
4           Total time: %{time_total}s\n
5           Download speed: %{speed_download} bytes/sec\n' \
6       -o /dev/null \
7       <file_server_ip>:<file_server_port>/bytes/<file_size_in_bytes>
```

The `'socks5'` flag is required to route the traffic through the Tor network, therefore the <tor_proxy> must be a tor client node. As mentioned before in subsection 4.4.2, we recommend this request to be performed in the client node's machine and to assign the value *'127.0.0.1:9000'* to <tor_proxy>. The `file_server_ip` and `file_server_port` refer to the web server and, as the name suggests, the `file_size_in_bytes` refers to the size of the file to be downloaded, in bytes. The used sizes were 51 200 bytes, 1 048 576 bytes 5 242 880 bytes, respectively the files used by Tor Metrics. The `-H 'Cache-Control: no-cache'` flag is used to prevent caching of the file, ensuring that each download request retrieves the file from the server rather than a cached version. The `-w` flag is used to format the output of the command, displaying the HTTP response code, for debugging purposes, time to first byte, to capture the latency, total time taken for the download, and download speed in bytes per second, referred onwards as throughput.

To collect data for the unobservability evaluation, we used the previous work of Pereira[47], by capturing traffic using `tcpdump` to *pcap* files and then using the captured traces to trains and evaluate machine learning models to simulate website fingerprinting attacks.

5.2 Performance Evaluation

To evaluate the performance of our solution, we conducted a series of experiments to measure the impact of the implemented features alone and in combination. In this section, we present the performance evaluation results, focusing on each metric. Firstly, we address the throughput results, followed by the total time taken to download files of different sizes, and finally, we analyze the latency results and the effect of the PPC feature on false cells and TLS packets count.

The experiments were performed with variable configurations for both features. The Packet Padding Cells feature was tested with different ϵ values, ranging from 0 to 5, with more collections between 0 and 1, variable onwards referred as ϵ_d. On the other hand, the Schedulers feature was also tested with different ϵ values, also ranging from 0 to 5, with more collections between 0 and 1, variable onwards referred as ϵ_j. Additionally, when individually testing the Schedulers, we used different mathematical distributions to generate jitter: *Poisson* and *Exponential*. We also tested both features together, with variable ϵ_d and ϵ_j values, using the *Poisson* distribution for jitter generation, due to time limitations of this work. These results are represented by heatmaps, with each ϵ as an axis, as the heat as the value of the metric.

In addition, we also present some results regarding the number of Packet Padding Cells generated during the experiments, compared with the TLS packets and total cells.

As mentioned earlier, the Tor Metrics project provides a continuous evaluation of the Tor network, including performance metrics such as throughput, total time to download files, and latency. In this chapter, we present our results and compare to the most recent results on Tor Metrics. The project provides results for 6 types of sources, but we focus on those which do not include the Conflux network, represented as mean of all medians. Conflux is a traffic splitting feature that Tor leverages to improve performance but, considering that our tests were conducted in a controlled and minimal network of relay, this features had no effect and must not be considered for results comparison.

All experiments were conducted in the environments described in section 5.1 and compared with control tests performed without the implemented features, for baseline performance measurement, and with the Tor Metrics measures at the time of writing. The values presented in the following sections were obtained by performing a download 100 times for each configuration and calculating the median, 10th and 90th percentiles.

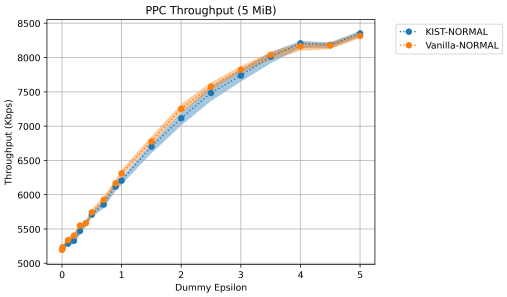
5.2.1 Throughput

Throughput is the unit of measurement that represents the amount of data flowing throw a certain point in the network. In our case, the throughput of the network is retrieved by downloading a file and measuring the download speed, in bytes per second. This is an important metric to evaluate the performance of our solution, as it directly impacts the user experience when browsing the web.

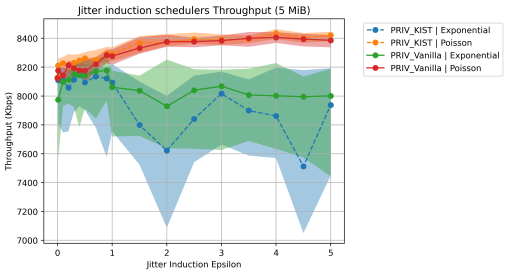
Below we share the throughput results obtained from our experiments and according to the above-mentioned test benches. We present the results for the local simulated environment in Figure 5.1 and the distributed environment in Figure 5.2. Each contains the corresponding results for each feature alone and both features combined, with the median represented as a thick line, together with a shaded area representing the first and third quartiles.

As shown in Figure 5.1 and Figure 5.2, the PPC feature has a more significant impact on throughput than the Schedulers feature. This impact has greater impact as ϵ_{PPC} decrease, reaches a plateau when ϵ_{PPC} is greater than 4. The distributed test bench showed more unstable results, but the overall trend is similar. The Schedulers feature has a smaller impact on throughput, with a slight decrease for ϵ_J values closer to 0. Unexpectedly, in the local emulated test bench, the Exponential distribution increases its variance on tests with ϵ_J greater than 1, which corresponded with a slight degradation in throughput. In all experiments, all types schedulers behaved similarly. For the experiments with both features enabled, the results emphasized the greater impact of the PPC feature on throughput. The heatmap clearly shows that the throughput only oscillate vertically, but remain consistent across different ϵ_J values.

As time of writing, Tor Metrics reports a median throughput value of 18 236 Mbps



(a) Only Packet Padding Cells Feature



(b) Only Jitter Injection Schedulers Feature

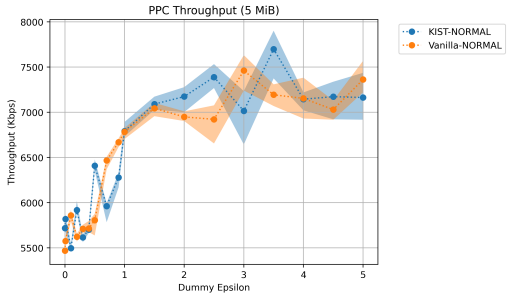
(c) Both Features

Figure 5.1: Throughput Results on Local Simulated Environment

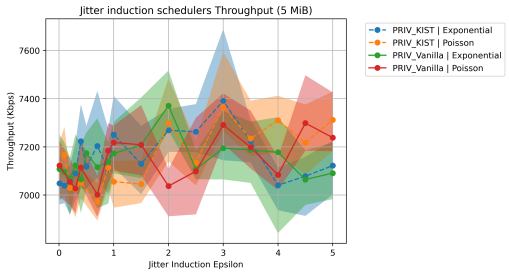
for files downloaded over the Tor network. Although, is important to note that these values are sampled from downloads of different files and only measure the throughput value between 4 MiB and 5 MiB of such files. In our case, we compare this value with the throughput values obtained from downloading files of 5 MiB.

Configuration	**Local**	**Distributed**
Tor Metrics	11 587	
Control	8 240 8 235	7 045 7 298
Only PPC	5 197 – 8 351	5 469 – 7 699
Only Jitter	7 512 – 8 435	6 973 – 7 391
PPC & Jitter	4 721 – 8 161	6 639 – 7 146

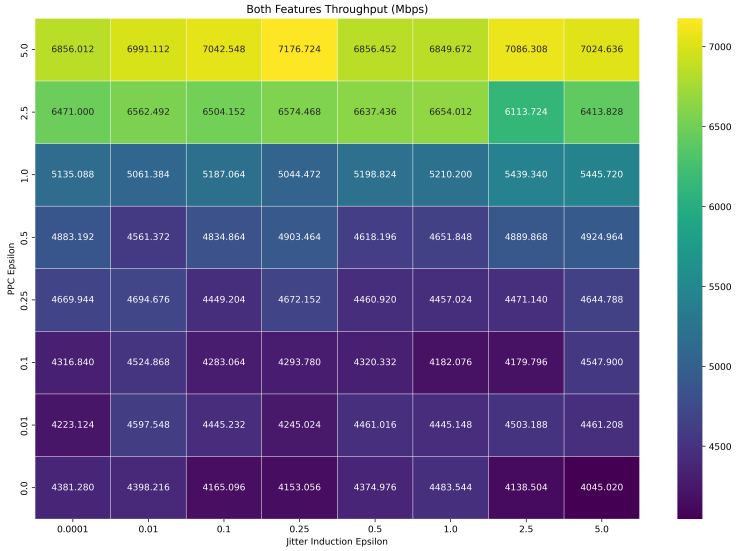
Table 5.1: Throughput Results Summary (Mbps)



(a) Only Packet Padding Cells Feature



(b) Only Jitter Injection Schedulers Feature



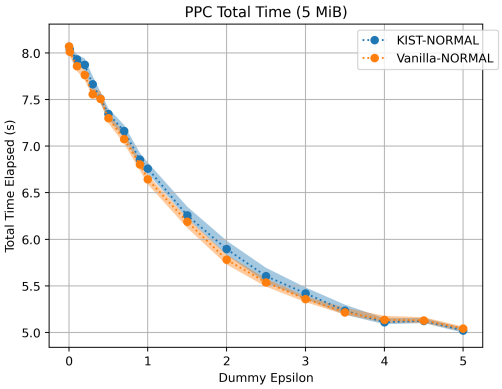
(c) Both Features

Figure 5.2: Throughput Results on Distributed Environment

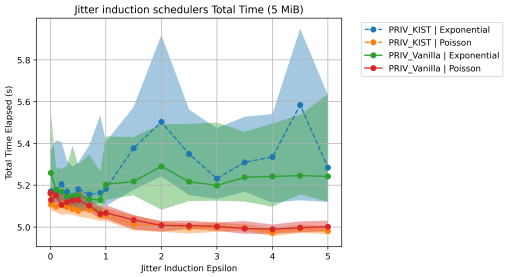
5.2.2 Total Time

Evaluating the total time necessary to download a file is also important to understand the effects of our solution on user experience and performance. Therefore, we present the total time results obtained from our experiments in Figure 5.3 for the local simulated environment and Figure 5.4 for the distributed environment. We followed the Tor Metrics directives and also present the results as a thick line which represents the median value, together with a shaded area representing the first and third quartiles.

As demonstrated in Figure 5.3 and Figure 5.4, the total time to download a file got results proportional to the throughput results, meaning that the PPC feature significantly influenced the total time to download. Regarding the Packet Padding Cells feature, the total time experienced is significantly lower as ϵ_{PPC} decreases. The same can be said about the Schedulers feature, although the impact is smaller. Additionally, the Exponential distribution showed a higher variance in the local test bench, and the median results unexpectably increased 0.1 seconds. This situation got normalized in the distributed test bench. Both schedulers behaved similarly in all experiments, thus we used the KIST variant and the Poisson distribution to test both features together. The results of the



(a) Only Packet Padding Cells Feature



(b) Only Jitter Injection Schedulers Feature

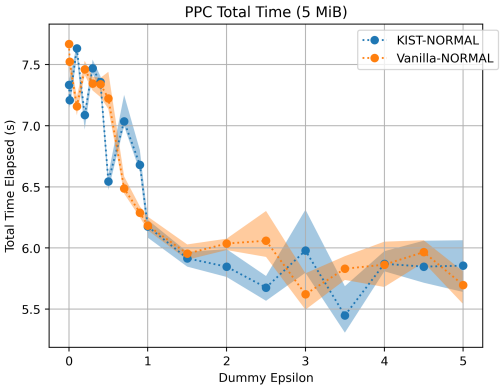
(c) Both Features

Figure 5.3: Total Time Results on Local Simulated Environment

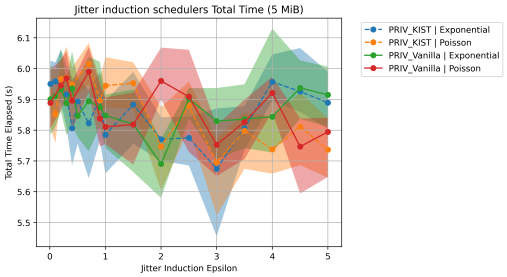
combined features also emphasized the greater impact of the PPC feature on the total time to download a file. Alike the throughput results, the total time heatmap proves that the schedulers feature has a negligible impact on the total time, when compared with the PPC.

Configuration	**Local**	**Distributed**
Tor Metrics	4.825	
Control	5.09 5.09	5.95 5.75
Only PPC	5.02 – 8.07	5.45 – 7.67
Only Jitter	5.58 – 4.97	5.67 – 6.01
PPC & Jitter	5.13 – 8.88	5.87 – 6.32

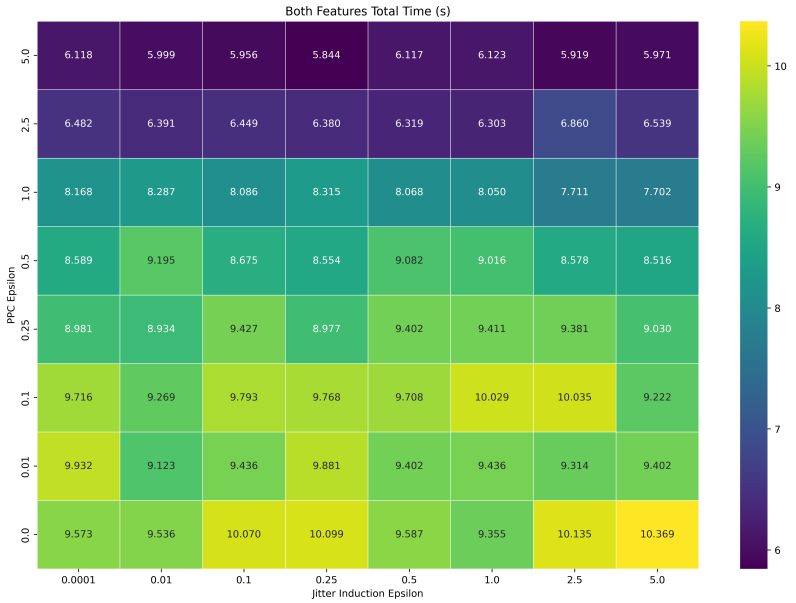
Table 5.2: Total Time Results Summary (s)



(a) Only Packet Padding Cells Feature



(b) Only Jitter Injection Schedulers Feature



(c) Both Features

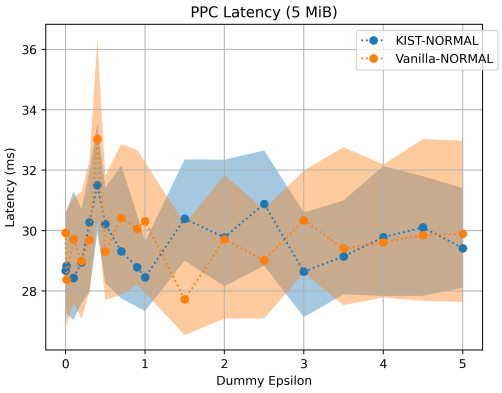
Figure 5.4: Total Time Results on Distributed Environment

5.2.3 Latency

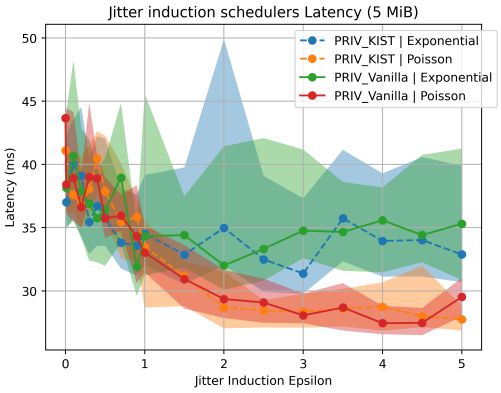
Tor Metrics defines their circuit round-trip latencies as the time between the HTTP request and the first byte of the HTTP response's header. This way, we obtained the latency results based on the time to the first byte of the HTTP response's header. This metric also represents a key aspect of a communication system such as Tor because it allows to understand the maximum rate information may be transmitted.

Similarly to the previous metrics, we present the latency results obtained from our experiments in Figure 5.5 for the local simulated environment and Figure 5.6 for the distributed environment. We followed the Tor Metrics directives and also present the results as a thick line which represents the median value, together with a shaded area representing the first and third quartiles.

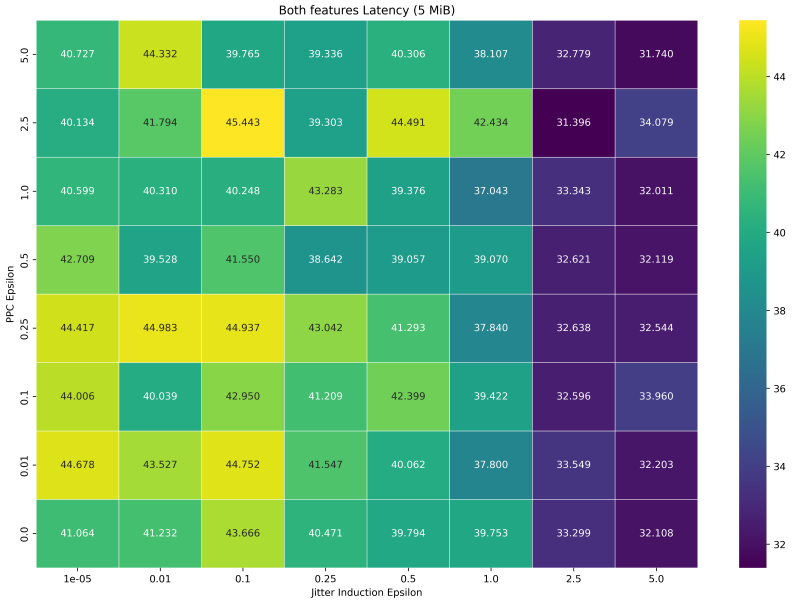
Unlike the previous metrics, the latency results are more sensitive to the schedulers jitter than to the PPC feature. As presented in Figure 5.5b and Figure 5.6b, the Packet



(a) Only Packet Padding Cells Feature



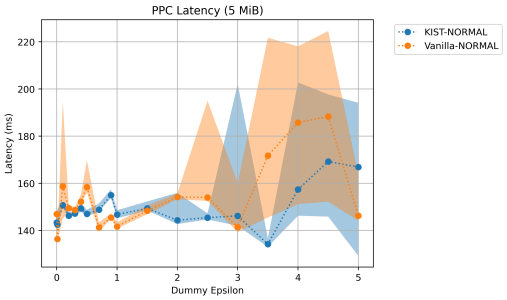
(b) Only Jitter Injection Schedulers Feature



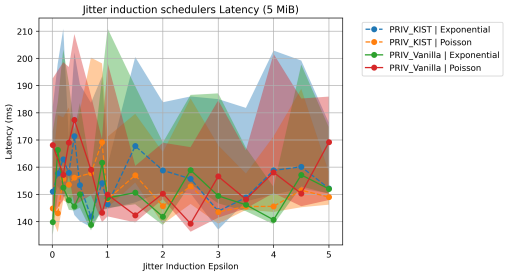
(c) Both Features

Figure 5.5: Latency Results on Local Simulated Environment

Padding Cells feature does not significantly impact the latency results, maintaining a relatively stable latency throughout the experiments. On the other hand, the Schedulers feature shows a more pronounced effect on latency, with noticeable increase when ϵ_J gets closer to 0, and reaching a minimum when ϵ_J is greater than 2. In the distributed environment, the latency results had a higher variance and a less noticeable increase in latency, as described before. As the time of conduction this evaluation, Tor Metrics reported a median latency of 280 ms. As reported in Table 5.3, the worse latency experience by our evaluation was 213.43 ms, with both features. This table also clearly shows that the impact of both features on latency are less apart as the impact differences produced in throughput and total time evaluations.



(a) Only Packet Padding Cells Feature



(b) Only Jitter Injection Schedulers Feature

(c) Both Features

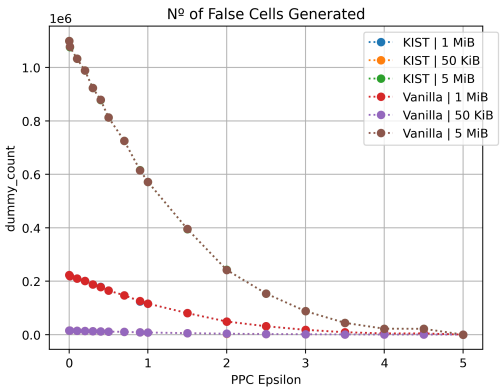
Figure 5.6: Latency Results on Distributed Environment

Configuration	Local	Distributed
Tor Metrics	280	
Control	30.59 30.53	181.36 192.54
Only PPC	27.45 – 33.03	134.16 – 188.30
Only Jitter	27.45 – 40.66	138.81 – 181.95
PPC & Jitter	31.40 – 45.44	170.37 – 213.43

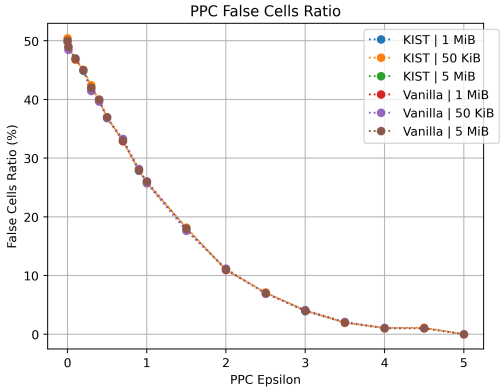
Table 5.3: Latency Results Summary (ms)

5.2.4 TLS Packets and Cells Analysis

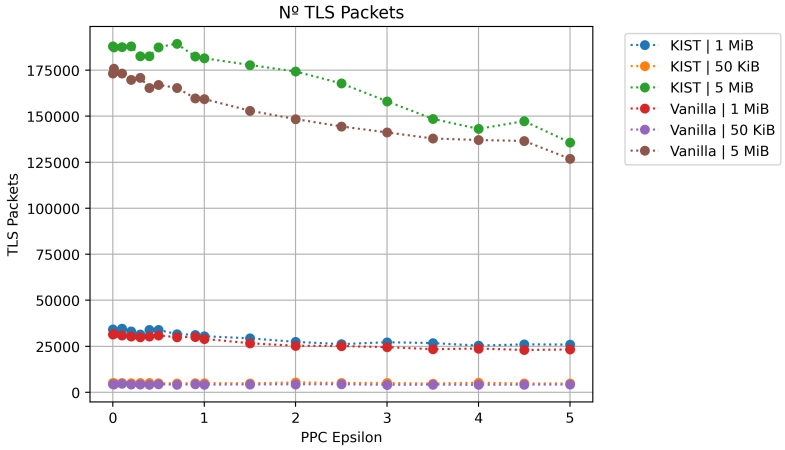
Finally, we present the results of our analysis on the number of false cells, the ratio of false cells, and the total number of TLS packets. These results are not directly comparable to the previous metrics, nor are addressed by Tor Metrics, but we consider relevant to demonstrate and validate the Packet Padding Cells generation and its direct influence in traffic shaping.



(a) Number of False Cells



(b) Ratio of False Cells



(c) Total TLS Packets

Figure 5.7: Cells and Packets Results

As expected, the number of false cells and their ratio to total cells decreases as the ϵ increases, with the highest ratio being 50% when $epsilon_d = 0$. However, we observed that the total number of TLS packets does not follow a directly proportional pattern to the number of false cells, as each TLS packet can contain multiple cells, thereby adding some randomness to the traffic metadata and shape.

5.3 Unobservability Evaluation

In this section, we evaluate the resistance of our system, focusing on the capability to withstand website fingerprinting attacks. As previously detailed in subsubsection 2.2.4.1,

this form of traffic analysis aims to identify the websites a user is visiting by analyzing encrypted traffic patterns to infer their online activity. The following subsections present the methods and tools used to perform our evaluation and the experimental observations.

5.3.1 Methods and Tools

To evaluate the resistance of our system against website fingerprinting attacks, we collected traces of website accesses to be used to train and test several machine learning models. We selected 100 websites from the Tranco Top 1M website list [37], and sampled each website 50 times, generating 5 000 samples per test configuration. The traces were captured using `tcpdump` to generate a *pcap* for each sample and for each circuit segment. The website access was simulated by executing `curl` requests to the list of websites. To perform these requests, we disabled caching to ensure that each request was treated as a new visit. The captured *pcap* files recorded all packet-level network activity, such as packet size, timing, and direction, thus used to evaluate resistance to fingerprinting attacks.

5.3.1.1 Machine Learning Models

With the before-mentioned traces and to perform the unobservability evaluation, we took inspiration in [47] work on the evaluation of website fingerprinting attacks resistance and employed a diverse set of machine learning models used in traffic analysis research. These models include Naïve Bayes, Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Random Forest, Extra Trees, Gradient Boosting, and XGBoost. These models were trained and tested using their default configurations without performing hyperparameter tuning to ensure consistency and provide a fair benchmark. They were trained using 80% of the collected data, leaving the other 20% for testing. Below we briefly summarize each of the used models:

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 neighbors 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.

5.3.1.2 Evaluation Metrics

With these models, we were able to train and test these machine learning algorithms using the collected traces, referred in subsection 5.3.1, to evaluate the resistance of our system against website fingerprinting attacks. To assess the effectiveness of our system in mitigating these attacks and of each machine learning model, we extracted several evaluation metrics such as:

Accuracy: Accuracy measures the ratio of correct predictions, True Positives (TP) and True Negatives (TN), to the total number of predictions, including False Positives (FP) and False Negatives (FN). This metric provides a general overview of the model's performance.

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

Precision: Precision measures the ratio of true positive to the total number of positives, important to understand the accuracy of positive predictions and the model's ability to avoid false positives.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall: Recall measures the ratio of true positives to the total number of actual positives, important to understand the model's ability to identify all relevant instances and avoid false negatives.

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1-Score: The F1-Score is the harmonic mean of precision and recall, providing a balanced measure that considers both metrics. It is particularly useful when the class distribution is imbalanced.

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

We consider a true positive if the model correctly identifies a website, a true negative if it correctly identifies that a non-website classes, a false positive if it incorrectly identifies a website, and a false negative if it fails to identify a website.

5.3.2 Experimental Observations

In this section, we present the results of our unobservability evaluation, focusing on the performance of the machine learning models in identifying websites based on the collected traces. We conducted experiments using different configurations of our system, varying the ϵ values for both the Packet Padding Cells feature and the Schedulers feature. To ease the understanding the performance of each model and each configuration, we present the results in tables and by metric, with 4 test scenarios: control (without any feature), only Packet Padding Cells feature with minimum ϵ value, only Schedulers feature with minimum ϵ value and both features with minimum ϵ values.

The first scenario is used to set a comparison between the already existing Tor software against the solution presented. The second and third scenarios were designed to isolate the impact of each feature on unobservability. The final scenario, which combines both features, allows for an analysis of their synergistic effects and an assessment of whether their interaction could inadvertently degrade the desired unobservability properties.

This way, we can better understand the impact of each feature on the resistance against website fingerprinting attacks, as well as the combined effect of both features.

To present the results, we split them by evaluation metric and with a bar plot, with a group of bars for each machine learning model, where the groups are composed by 4 bars, corresponding to the used test scenarios.

5.3.2.1 Accuracy

The bar plot in Figure 5.8 presents the performance of all models under the four test scenarios. The `Packet Padding Cells` feature alone presented the overall highest values, specially under the `XGBoost` model. On the other hand, both scenarios with only the modified schedulers and with both features, showed very similar results across all models, demonstrating the effectiveness of the Jitter Induction Mechanism over Tor Schedulers and that the features do not negatively impact each other when combined, even though some models achieved better accuracy on both features than only the modified scheduler. The most accurate models were the `XGBoost`, `Random Forest` and `Extra Trees`.

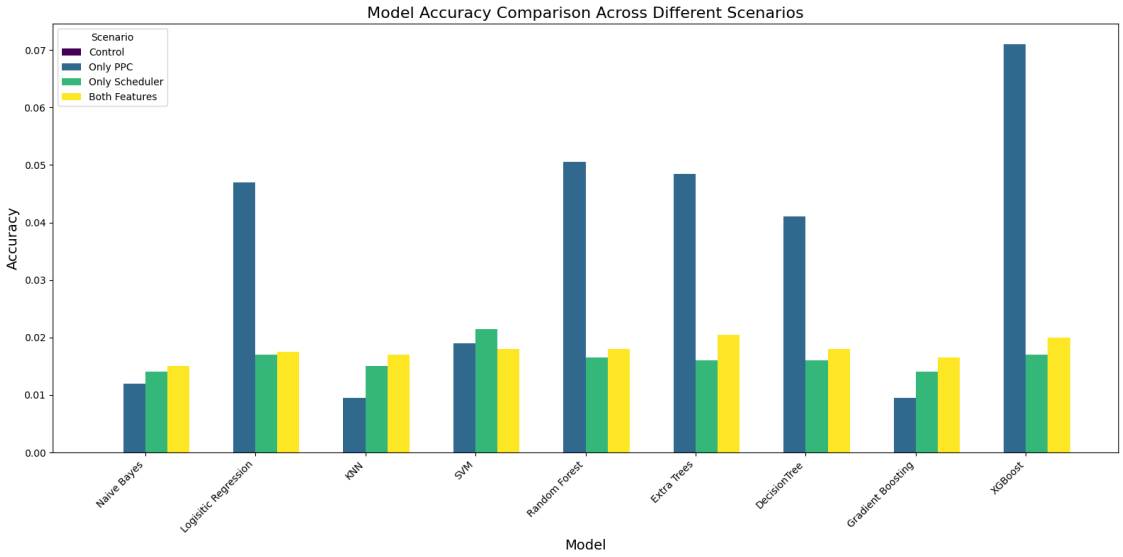


Figure 5.8: Comparison of models' accuracy

5.3.2.2 Precision

The bar plot in Figure 5.9 shows the precision of Machine Learning models over the referred testing scenarios. The results were tightly coupled with the accuracy results, were the most accurate models correspond to the most precise, specially over the Packet Padding Cells scenario. Once again, the higher precision over this scenario proves it to be the less effective against machine learning algorithms. Moreover, the other scenarios show both similar and lower precision values. The most precise models were the `XGBoost`, `Extra Trees` and `Logistic Regression`

5.3.2.3 Recall

As shown in Figure 5.10, the recall results are very similar to the accuracy across all models scenarios, which can indicate that models' performance is close to random guessing. Akin to accuracy, the best model is `XGBoost`, followed by `Random Forest` and `Extra Trees`, and the `Packet Padding Cells` scenario proves to have the highest values of recall.

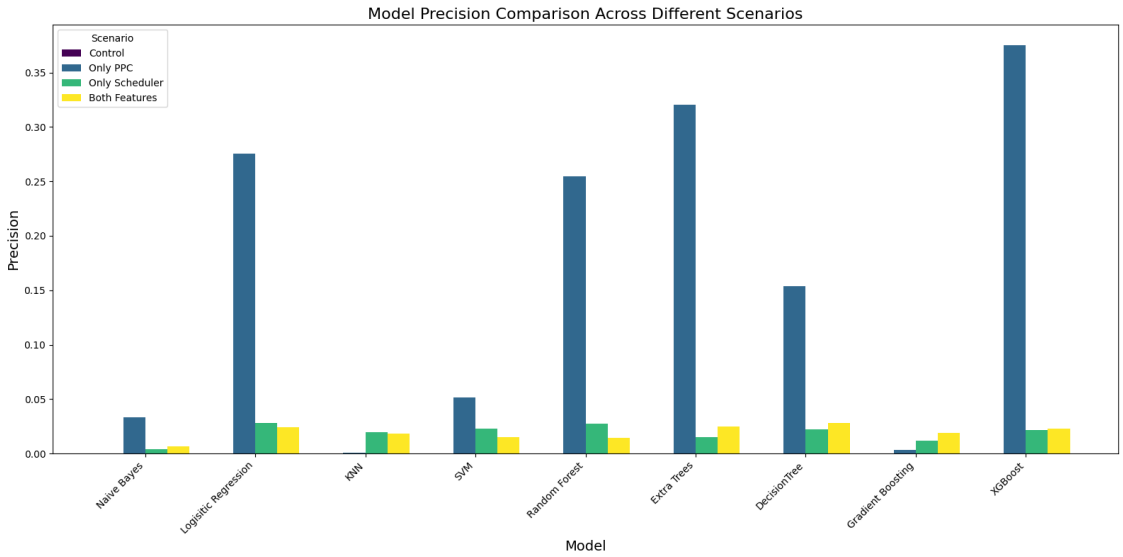


Figure 5.9: Comparison of models' precision

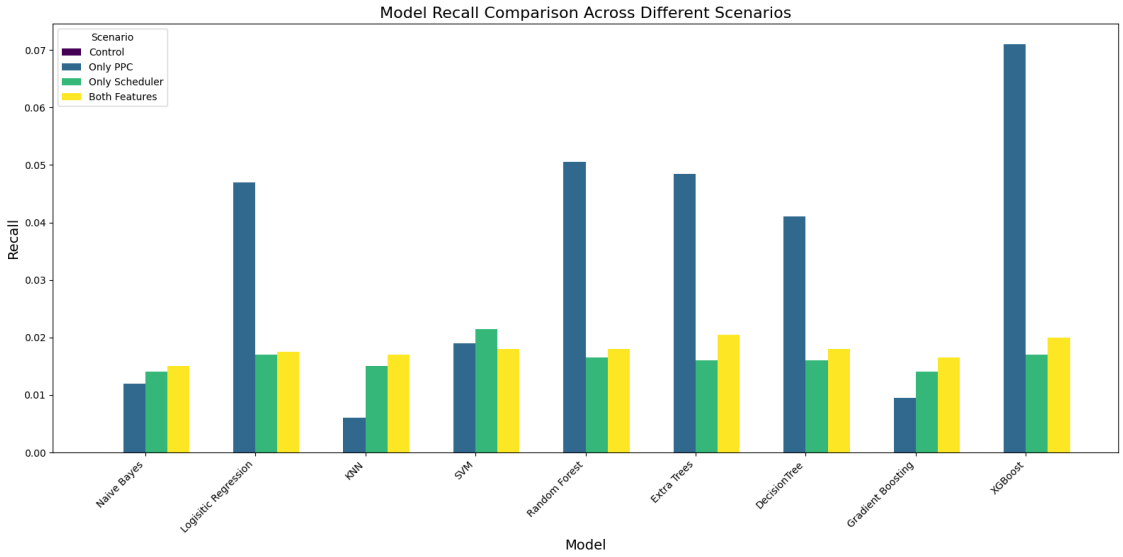


Figure 5.10: Comparison of models' recall

5.3.2.4 F1-Score

Finally, the bar plot in Figure 5.11 demonstrates the results of F1-Score of the experiments. Similarly to the previous metrics, the `Packet Padding Cells` feature achieved higher values compared with the other test cases. The best performant models, regarding F1-Score, were the `XGBoost`, `Random Forest` and `Extra Trees`.

5.4 Formal Validation

As previously mentioned in section 2.4, our approach leverages Differential Privacy principles to enhance the anonymity of users within the Tor network. In this section,

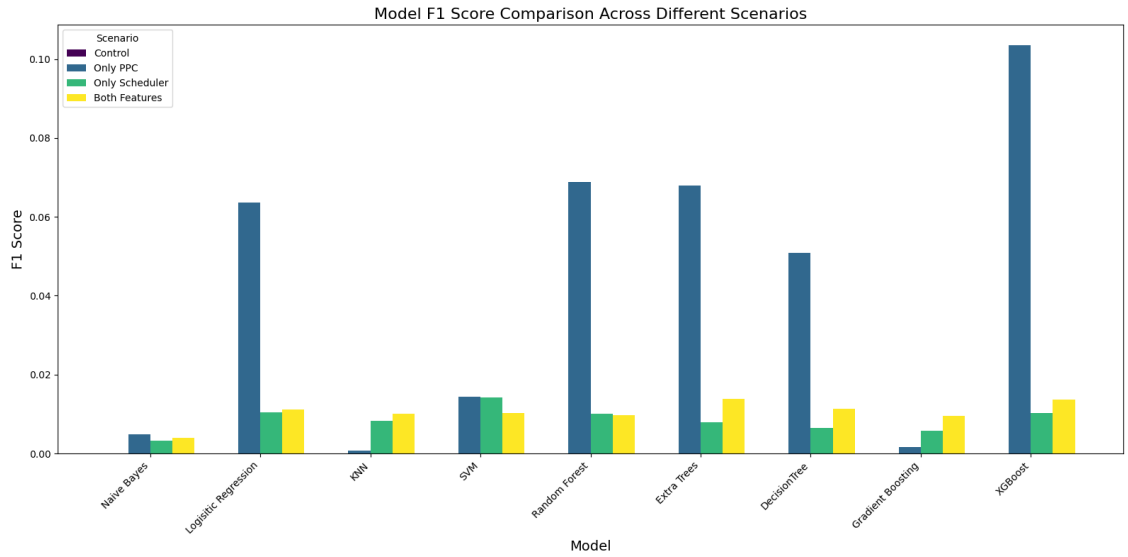


Figure 5.11: Comparison of models' F1-Score

we provide a formal validation of our solution by redefining Differential Privacy in the context of our system and relating its theorems to our proposed mechanisms: size sets and delta sets.

An algorithm satisfies Differential Privacy if, for any two adjacent datasets (differing by a single element), the probability of any output does not change significantly when one element is added or removed. This algorithm uses a randomized function, here denoted as F, and its output will be similar for both datasets, with a bound defined by the privacy parameter ϵ. This parameter provides a knob to tune the amount of privacy the definition provides.

Our solution introduces two key mechanisms to achieve Differential Privacy: Packet Padding Cells, which vary the size of TLS packets, and the Jitter Injection Schedulers, which vary the time intervals between TLS Packets. This way, we consider two datasets as adjacent if they differ by a single packet size or a single time interval, respectively.

The first mechanism, Packet Padding Cells, modifies the size of TLS packets by adding dummy data, which provides irreversible privacy to the TLS packets sizes, through the Post Processing property, thereby creating a set of possible packet sizes (size sets). This feature ensures that the probability of observing a particular packet size remains similar, regardless of whether a specific packet is present or absent in the dataset, with the use of a randomized response algorithm, such as Listing 4.1. More formally, for any two adjacent datasets D and D', and for any possible output S (a specific packet size), the following condition holds:

$$P(F(D) \in S) \leq e^{\epsilon} \cdot P(F(D') \in S)$$

In this context, F represents the randomized response function that adds padding to packets, which satisfies ϵ-Differential Privacy.

The second mechanism, Jitter Injection Schedulers, introduces variability in the timing of packet transmissions by adding random delays (delta sets). This feature leverages mathematical distributions, such as Listing 5.1 to generate differential private time intervals, ensuring that the timing of packet transmissions does not reveal information about the presence or absence of specific packets. Formally, for any two adjacent datasets D and D', and for any possible output T (a specific time interval), the following condition holds:

$$P(F(D) \in T) \leq e^{\epsilon} \cdot P(F(D') \in T)$$

```c
int poisson_mechanism(double epsilon, int target, int lower, int upper)
{
  const double lambda = (double)(upper - lower) / epsilon;

  double p = exp(-lambda);
  double sum = p;
  int k = 0;

  while (sum < uniform()) {
      k++;
      p *= lambda / k;
      sum += p;
  }

  return CLAMP(lower, target + k, upper);
}
```

Listing 5.1: Poisson Distribution Pseudo-Random Number Generator implementation.

In this context, F represents the randomized function that adds jitter to packet transmissions. Moreover, given f as the function that givens default time interval between packets, we can define F as:

$$F(D) = f(D) + \text{Poisson}(\lambda)$$

where $\text{Poisson}(\lambda)$ is a random variable drawn from a Poisson distribution with parameter λ.

Both features most be carefully used to ensure minimal privacy loss, as the total privacy cost of multiple differentially private mechanisms is the sum of their individual privacy costs, according to the Sequential Composition theorem. Therefore, if both features are applied to the same dataset, the overall privacy guarantee is given by:

$$P(F(D) \in G) \leq e^{\epsilon_1 + \epsilon_2} \cdot P(F(D') \in G)$$

where ϵ_1 and ϵ_2 are the privacy parameters for the Packet Padding Cells and Jitter Injection Schedulers, respectively, and G is any possible output (a specific packet size and time interval).

5.5 Summary

This chapter presented a comprehensive experimental validation and evaluation of the TTT prototype, focusing on its performance and unobservability under various configurations and conditions, and on the formal validation of the Differential Privacy guarantees provided by the prototype. Our analysis unrevealed significant insights over the extension of Tor source code and the impact of both jitter injection and packet padding based on Differential Privacy over both performance and unobservability, as well as the software privacy guarantees provided to users.

To test the performance of both implemented features, we run several tests with variable values of ϵ and mathematical distributions, collecting the latency, throughput and total time of each measurement. Furthermore, we also collected data over the number of false cells generated, the ratio of these cells compared to the real ones and the effect on TLS packets number. The tests demonstrated that the PPC feature impacts total time and throughput as ϵ becomes closer to 0, regardless the latency remains significantly altered. On the other hand, the Jitter Induction Schedulers feature showed that latency increases when ϵ becomes closer to 0, even though throughput and total time only increase slightly. When both features are combined, it is clear that the latter feature does not impact performance as much as the PPC. Regarding the values of both ϵ of each feature, results became unaltered when the variable was bigger than 4. The collected measures over the number of false cells confirmed that the creation of these types of cells increases TLS on a random and non-regular distribution, altering and randomizing traffic characteristics against machine learning algorithms.

Regarding the unobservability evaluation, we collected 5 000 website traces for each test scenario and used such traces to train and evaluate a set of 9 machine learning models to simulate website fingerprinting attacks. In total, we considered 4 test scenarios: a control scenario without any implemented feature, a PPC scenario only with this feature and another Scheduler scenario only with the modified schedulers, and finally a scenario with both features. The results showed that the PPC feature provided less unobservability compared to using only the scheduler or both features, and combining both features did not reduce their individual effectiveness.

The formal validation demonstrated that Packet Padding Cells and Jitter Induction Schedulers features modify the TLS packets size and time intervals between packets, respectively, which contain Differential Privacy properties. The PPC features is able to provide such properties due to a randomized response algorithm and by generating false cells and altering the TLS packet sizes between the different circuit segments. The Jitter Induction Schedulers were able to provide these properties by using mathematical distributions to generate pseudo-random numbers to define the time intervals between TLS packets over the different circuit segments. Nevertheless, users must account for privacy loss. The long use of these features can reduce the privacy guarantees to eventually none.

In conclusion, the experimental evaluation and validation demonstrate that the proposed solution effectively strengthens Tor's unobservability properties while ensuring Differential Privacy guarantees, without incurring performance degradation severe enough to render the system impractical.

6

Conclusions

6.1 Main Contributions

Our main contribution is the extension of Tor source code with 2 independent features focused on enhancing its unobservability capabilities. The prototype can be used by the scientific community and practitioners to investigate the developed features and extend the solution with more mathematical distributions and Differential Privacy algorithms, for example.

Also, Tor network users can deploy their relays or use the developed solution in the already existing Tor network, as the solution is fully compactible. Our additional contributions include (1) a comprehensive analysis of TTT's performance in more than 170 testing configurations on an emulated local Tor network and over 170 testing configurations on a distributed Tor network, assessing latency, throughput and total time conditions over multiple different real-world conditions; (2) a detailed study of the performance of 9 machine learning models and the resistance of the prototype against website fingerprinting attacks; (3) formal validation of the Differential Privacy properties of the developed prototype and the implications of such properties in the Tor network, for both their users and hosts; and (4) a strong solution that enhances Tor software and network fingerprinting resistance with better privacy guarantees for its users, through the addition of a randomized response algorithm to generate false cells and the adaption of 2 Tor schedulers to use mathematical distributions to add jitter condition and randomly modify the time intervals between each packet.

6.2 Future Work

The work developed in this thesis opens several promising avenues for future research and development. Potential directions include, but are not limited to, the following:

- **Integration with the Tor project**: Incorporating the proposed solution into the official Tor source codebase in order to validate its utility and assess its long-term impact within the broader Tor community.

- **Broader unobservability testing**: Conducting unobservability evaluations under a wider range of configurations to gain deeper insights into which settings yield the most robust protection.
- **Extension of the Packet Padding Cells feature**: Enhancing the current implementation to generate padding cells not only after receiving a Tor cell, but also during periods of inactivity. This modification has the potential to further strengthen the system's Differential Privacy guarantees.
- **Evaluation in realistic distributed environments**: Performing experimental validation on a geographically distributed test bench with relays spanning multiple continents, thereby approximating real-world network conditions more closely.
- **Expansion of machine learning-based unobservability experiments**: Extending the experimental framework to encompass a broader variety of machine learning algorithms and significantly larger datasets, in order to obtain more comprehensive and generalizable results.

Bibliography

[1] L. von Ahn, A. Bortz, and N. J. Hopper. "k-anonymous message transmission". In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. CCS '03. Washington D.C., USA: Association for Computing Machinery, 2003, 122–130. ISBN: 1581137389. DOI: 10.1145/948109.948128. URL: https://doi.org/10.1145/948109.948128 (cit. on p. 14).

[2] M. Alimardani and S. Milan. "The Internet as a Global/Local Site of Contestation: The Case of Iran". In: *Global Cultures of Contestion*. Ed. by D. T. Della and G. K. Thomas. Springer, 2018, pp. 171–192. URL: https://link.springer.com/chapter/10.1007/978-3-319-78889-7_9 (cit. on pp. 1, 3).

[3] S. Aryan, H. Aryan, and J. A. Halderman. "Internet Censorship in Iran: A First Look". In: *Proceedings of the 3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI 13)*. USENIX Association, 2013. URL: https://www.usenix.org/conference/foci13/workshop-program/presentation/aryan (cit. on pp. 1, 3).

[4] L. Barman et al. "Groove: Flexible Metadata-Private Messaging". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, 2022-07, pp. 735–750. ISBN: 978-1-939133-28-1. URL: https://www.usenix.org/conference/osdi22/presentation/barman (cit. on p. 6).

[5] 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)*. ACM, 2020, pp. 35–48. DOI: 10.1145/3372297.3423345. URL: https://doi.org/10.1145/3372297.3423345 (cit. on p. 2).

[6] A. Campan and T. Truta. "Data and Structural k-Anonymity in Social Networks". In: vol. 5456. 2008-01, pp. 33–54. ISBN: 978-3-642-01717-9. DOI: 10.1007/978-3-642-01718-6_4 (cit. on p. 15).

[7] B. Carvalho. "TIRDEANON. DEANONYMIZATION AND TRAFFIC-UNOBSERVABILITY ANALYSIS OF TIR AS A STRENGTHENED SOLUTION TO IMPROVE ANONYMITY GUARANTEES IN TOR". MA thesis. NOVA School of Science and Technology, 2023 (cit. on p. 1).

[8] S. Chakravarty et al. "On the Effectiveness of Traffic Analysis against Anonymity Networks Using Flow Records". In: *Passive and Active Measurement*. Ed. by M. Faloutsos and A. Kuzmanovic. Cham: Springer International Publishing, 2014, pp. 247–257 (cit. on pp. 1–3, 13).

[9] D. Chaum. "Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms". In: *Secure Electronic Voting*. Ed. by D. A. Gritzalis. Boston, MA: Springer US, 2003, pp. 211–219. ISBN: 978-1-4615-0239-5. DOI: 10.1007/978-1-4615-0239-5_14. URL: https://doi.org/10.1007/978-1-4615-0239-5_14 (cit. on p. 5).

[10] 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, 2022-08, pp. 753–770. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/cherubin (cit. on pp. 1, 2).

[11] H. Corrigan-Gibbs, D. Boneh, and D. Mazieres. "Riposte: An anonymous messaging system handling millions of users". In: *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. 2015, pp. 321–338 (cit. on p. 2).

[12] 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 p. 6).

[13] G. Danezis. "Statistical Disclosure Attacks". In: *Security and Privacy in the Age of Uncertainty*. Ed. by D. Gritzalis et al. Boston, MA: Springer US, 2003, pp. 421–426. ISBN: 978-0-387-35691-4 (cit. on p. 2).

[14] A. Danial. *cloc*. Version v2.06. URL: https://github.com/AlDanial/cloc (cit. on p. 34).

[15] R. Dingledine, N. Mathewson, and P. Syverson. "Tor: The Second-Generation Onion Router". In: *Proceedings of the 13th USENIX Security Symposium*. USENIX Association, 2004, pp. 303–320. URL: https://www.usenix.org/conference/usenixsecurity04/tor-second-generation-onion-router (cit. on pp. 2, 3, 6, 9, 13, 23, 24).

[16] C. Dwork. "Differential privacy". In: *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*. ICALP'06. Venice, Italy: Springer-Verlag, 2006, 1–12. ISBN: 3540359079. DOI: 10.1007/11787006_1. URL: https://doi.org/10.1007/11787006_1 (cit. on pp. 2, 15).

[17] C. Dwork and A. Roth. "The Algorithmic Foundations of Differential Privacy". In: *Found. Trends Theor. Comput. Sci.* 9.3–4 (2014-08), 211–407. ISSN: 1551-305X. DOI: 10.1561/0400000042. URL: https://doi.org/10.1561/0400000042 (cit. on pp. 15–17).

[18] C. Dwork et al. "Calibrating noise to sensitivity in private data analysis". In: *Proceedings of the Third Conference on Theory of Cryptography*. TCC'06. New York, NY: Springer-Verlag, 2006, 265–284. ISBN: 3540327312. DOI: 10.1007/11681878_14. URL: https://doi.org/10.1007/11681878_14 (cit. on pp. 2, 15, 16).

[19] Úlfar Erlingsson, V. Pihur, and A. Korolova. "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response". In: *Proceedings of the 21st ACM Conference on Computer and Communications Security*. Scottsdale, Arizona, 2014. URL: https://arxiv.org/abs/1407.6981 (cit. on p. 17).

[20] G. Figueira, D. Barradas, and N. Santos. "Stegozoa: Enhancing WebRTC Covert Channels with Video Steganography for Internet Censorship Circumvention". In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '22. Nagasaki, Japan: Association for Computing Machinery, 2022, 1154–1167. ISBN: 9781450391405. DOI: 10.1145/3488932.3517419. URL: https://doi.org/10.1145/3488932.3517419 (cit. on p. 2).

[21] D. Goldschlag, M. Reed, and P. Syverson. "Hiding Routing Information". In: 1996-08. ISBN: 978-3-540-61996-3. DOI: 10.1007/3-540-61996-8_37 (cit. on pp. 5, 6).

[22] A. Gosain and N. Chugh. "Privacy Preservation in Big Data". In: *International Journal of Computer Applications* 100 (2014-08), pp. 44–47. DOI: 10.5120/17619-8322 (cit. on p. 15).

[23] J. van den Hooff et al. "Vuvuzela: scalable private messaging resistant to traffic analysis". In: *Proceedings of the 25th Symposium on Operating Systems Principles*. SOSP '15. Monterey, California: Association for Computing Machinery, 2015, 137–152. ISBN: 9781450338349. DOI: 10.1145/2815400.2815417. URL: https://doi.org/10.1145/2815400.2815417 (cit. on p. 6).

[24] N. Hopper and E. Y. Vasserman. "On the effectiveness of k;-anonymity against traffic analysis and surveillance". In: *Proceedings of the 5th ACM Workshop on Privacy in Electronic Society*. WPES '06. Alexandria, Virginia, USA: Association for Computing Machinery, 2006, 9–18. ISBN: 1595935568. DOI: 10.1145/1179601.1179604. URL: https://doi.org/10.1145/1179601.1179604 (cit. on pp. 2, 15).

[25] M. Horta. "TOR K-ANONYMITY AGAINST DEEP LEARNING WATERMARKING ATTACKS: validating a Tor k-Anonymity input circuit enforcement against a deep learning watermarking attack". MA thesis. NOVA School of Science and Technology, 2022-10 (cit. on p. 1).

[26] A. Houmansadr et al. "IP over Voice-over-IP for censorship circumvention". In: *CoRR* abs/1207.2683 (2012). arXiv: 1207.2683. URL: http://arxiv.org/abs/1207.2683 (cit. on p. 2).

[27] A. Inc. *Learning with Privacy at Scale*. URL: https://machinelearning.apple.com/research/learning-with-privacy-at-scale (cit. on p. 17).

[28] M. Jakobsson. "Flash mixing". In: *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '99. Atlanta, Georgia, USA: Association for Computing Machinery, 1999, 83–89. ISBN: 1581130996. DOI: 10.1145/301308.301333. URL: https://doi.org/10.1145/301308.301333 (cit. on p. 5).

[29] M. Jakobsson and A. Juels. "An optimally robust hybrid mix network". In: *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*. PODC '01. Newport, Rhode Island, USA: Association for Computing Machinery, 2001, 284–292. ISBN: 1581133839. DOI: 10.1145/383962.384046. URL: https://doi.org/10.1145/383962.384046 (cit. on p. 5).

[30] R. Jansen and N. Hopper. "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation". In: *Network and Distributed System Security Symposium*. 2012 (cit. on pp. 12, 33).

[31] R. Jansen, T. Vaidya, and M. Sherr. "Point Break: A Study of Bandwidth Denial-of-Service Attacks against Tor". In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, 2019-08, pp. 1823–1840. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/jansen (cit. on pp. 1–3).

[32] R. Jansen et al. "KIST: Kernel-Informed Socket Transport for Tor". In: *ACM Trans. Priv. Secur.* 22.1 (2018-12). ISSN: 2471-2566. DOI: 10.1145/3278121. URL: https://doi.org/10.1145/3278121 (cit. on pp. 3, 11, 12, 33).

[33] A. Jerichow et al. "Real-time mixes: a bandwidth-efficient anonymity protocol". In: *IEEE J.Sel. A. Commun.* 16.4 (2006-09), 495–509. ISSN: 0733-8716. DOI: 10.1109/49.668973. URL: https://doi.org/10.1109/49.668973 (cit. on p. 6).

[34] A. Johnson et al. "Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries". In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2013. DOI: 10.1145/2508859.2516651. URL: https://doi.org/10.1145/2508859.2516651 (cit. on pp. 1, 3).

[35] D. Kesdogan, J. Egner, and R. Büschkes. "Stop- and- Go-MIXes Providing Probabilistic Anonymity in an Open System". In: vol. 1525. 1998-04, pp. 83–98. ISBN: 978-3-540-65386-8. DOI: 10.1007/3-540-49380-8_7 (cit. on p. 7).

[36] D. Lazar, Y. Gilad, and N. Zeldovich. "Karaoke: Distributed private messaging immune to passive traffic analysis". In: *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2018, pp. 711–725 (cit. on pp. 2, 6).

[37] V. Le Pochat et al. "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation". In: *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. NDSS 2019. 2019-02. DOI: 10.14722/ndss.2019.23386 (cit. on p. 49).

[38] 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).

[39] N. Mathewson and R. Dingledine. "Practical Traffic Analysis: Extending and Resisting Statistical Disclosure". In: *Privacy Enhancing Technologies*. Ed. by D. Martin and A. Serjantov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 17–34. ISBN: 978-3-540-31960-3 (cit. on p. 2).

[40] S. Matic, C. Troncoso, and J. Caballero. "Dissecting Tor Bridges: A Security Evaluation of their Private and Public Infrastructures". In: *Network and Distributed System Security Symposium*. 2017. URL: https://api.semanticscholar.org/CorpusID:28430784 (cit. on p. 10).

[41] U. Moeller. "Mixmaster Protocol Version 2". In: 2004. URL: https://api.semanticscholar.org/CorpusID:215824742 (cit. on p. 6).

[42] M. Nasr, A. Bahramali, and A. Houmansadr. "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning". In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. Toronto, ON, Canada: ACM, 2018, pp. 1962–1976. DOI: 10.1145/3243734.3243858 (cit. on pp. 1, 13).

[43] J. P. Near and C. Abuah. *Programming Differential Privacy*. Vol. 1. 2021. URL: https://programming-dp.com/ (cit. on pp. 2, 15–17).

[44] N. Nipane, I. Dacosta, and P. Traynor. ""Mix-in-Place" anonymous networking using secure function evaluation". In: *Proceedings of the 27th Annual Computer Security Applications Conference*. ACSAC '11. Orlando, Florida, USA: Association for Computing Machinery, 2011, 63–72. ISBN: 9781450306720. DOI: 10.1145/2076732.2076742. URL: https://doi.org/10.1145/2076732.2076742 (cit. on p. 6).

[45] 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)*. 2022, pp. 1915–1932. DOI: 10.1109/SP46214.2022.9833801 (cit. on pp. 1, 13).

[46] M. Ohkubo and M. Abe. "A Length-Invariant Hybrid Mix". In: *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*. Vol. 1976. Lecture Notes in Computer Science. Springer, 2000, pp. 178–191. DOI: 10.1007/3-540-44448-3_14 (cit. on p. 5).

[47] H. Pereira. "MIRACE: Multi-Path Integrated Routing Architecture for Censorship Evasion". MA thesis. NOVA School of Science and Technology, 2024 (cit. on pp. 2, 8, 40, 49).

[48] A. M. Piotrowska et al. "The Loopix Anonymity System". In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017-08, pp. 1199–1216. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska (cit. on pp. 2, 7).

[49] T. Project. *Circumvention*. URL: https://tb-manual.torproject.org/circumvention/ (cit. on pp. 2, 10).

[50] T. Project. *Tor Plugable Transports*. URL: https://2019.www.torproject.org/docs/pluggable-transports (cit. on pp. 2, 10).

[51] M. S. Rahman et al. "Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks". In: *Proceedings on Privacy Enhancing Technologies* 2020.3 (2020), pp. 5–24. DOI: 10.2478/popets-2020-0032 (cit. on pp. 1, 2).

[52] F. Rezaei and A. Houmansadr. "FINN: Fingerprinting Network Flows using Neural Networks". In: *Proceedings of the 37th Annual Computer Security Applications Conference* (2021). URL: https://api.semanticscholar.org/CorpusID:243797821 (cit. on p. 1).

[53] A. Sabzi et al. "NetShaper: A Differentially Private Network Side-Channel Mitigation System". In: *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, 2024-08, pp. 3385–3402. ISBN: 978-1-939133-44-1. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/sabzi (cit. on pp. 2, 18).

[54] T. Shen et al. " DAENet: Making Strong Anonymity Scale in a Fully Decentralized Network ". In: *IEEE Transactions on Dependable and Secure Computing* 19.04 (2022-07), pp. 2286–2303. ISSN: 1941-0018. DOI: 10.1109/TDSC.2021.3052831. URL: https://doi.ieeecomputersociety.org/10.1109/TDSC.2021.3052831 (cit. on p. 2).

[55] 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 (cit. on p. 3).

[56] V. Shmatikov and M.-H. Wang. "Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses". In: vol. 4189. 2006-01, pp. 18–33. ISBN: 978-3-540-44601-9. DOI: 10.1007/11863908_2 (cit. on pp. 2, 9).

[57] 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, 1928–1943. ISBN: 9781450356930. DOI: 10.1145/3243734.3243768. URL: https://doi.org/10.1145/3243734.3243768 (cit. on pp. 1, 2, 13).

[58] P. Sirinam et al. "Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, 1131–1148. ISBN: 9781450367479. DOI: 10.1145/3319535.3354217. URL: https://doi.org/10.1145/3319535.3354217 (cit. on p. 13).

[59] Y. Sun et al. "RAPTOR: routing attacks on privacy in tor". In: *Proceedings of the 24th USENIX Conference on Security Symposium*. SEC'15. Washington, D.C.: USENIX Association, 2015, 271–286. ISBN: 9781931971232 (cit. on p. 13).

[60] L. Sweeney. "k-anonymity: a model for protecting privacy". In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 10.5 (2002-10), 557–570. ISSN: 0218-4885. DOI: 10.1142/S0218488502001648. URL: https://doi.org/10.1142/S0218488502001648 (cit. on pp. 2, 14).

[61] L. Sweeney. "Simple Demographics Often Identify People Uniquely". In: *Carnegie Mellon University, Data Privacy* (2000). URL: http://dataprivacylab.org/projects/identifiability/ (cit. on p. 14).

[62] C. Tang and I. Goldberg. "An improved algorithm for tor circuit scheduling". In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. Chicago, Illinois, USA: Association for Computing Machinery, 2010, 329–339. ISBN: 9781450302456. DOI: 10.1145/1866307.1866345. URL: https://doi.org/10.1145/1866307.1866345 (cit. on pp. 11, 12).

[63] N. Tyagi et al. "Stadium: A Distributed Metadata-Private Messaging System". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. SOSP '17. Shanghai, China: Association for Computing Machinery, 2017, 423–440. ISBN: 9781450350853. DOI: 10.1145/3132747.3132783. URL: https://doi.org/10.1145/3132747.3132783 (cit. on pp. 2, 6, 8).

[64] A. Vilalonga, J. S. Resende, and H. Domingos. *TorKameleon: Improving Tor's Censorship Resistance with K-anonymization and Media-based Covert Channels*. 2023. arXiv: 2303.17544 [cs.CR]. URL: https://arxiv.org/abs/2303.17544 (cit. on p. 2).

[65] A. Vilalonga et al. "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: *INForum* (2024). URL: https://www.inforum.pt/static/files/papers/INForum_2024_paper_77.pdf (cit. on pp. 2, 18).

[66] M. Wang et al. "Leveraging strategic connection migration-powered traffic splitting for privacy". In: *Proceedings on Privacy Enhancing Technologies* 2022.3 (2022-07), 498–515. ISSN: 2299-0984. DOI: 10.56553/popets-2022-0083. URL: http://dx.doi.org/10.56553/popets-2022-0083 (cit. on p. 2).

[67] S. L. Warner. "Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias". In: *Journal of the American Statistical Association* 60.309 (1965). PMID: 12261830, pp. 63–69. DOI: 10.1080/01621459.1965.10480775. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1965.10480775 (cit. on p. 17).

[68] Z. Weinberg et al. "StegoTorus: A camouflage proxy for the Tor anonymity system". In: 2012-10, pp. 109–120. DOI: 10.1145/2382196.2382211 (cit. on p. 2).

[69] P. Winter and S. Lindskog. "How the Great Firewall of China is Blocking Tor". In: *Proceedings of the 2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI)*. USENIX Association, 2012. URL: https://www.usenix.org/conference/foci12/workshop-program/presentation/winter (cit. on pp. 1–3).

[70] D. I. Wolinsky et al. "Dissent in Numbers: Making Strong Anonymity Scale". In: *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX Association, 2012-10, pp. 179–182. ISBN: 978-1-931971-96-6. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/wolinsky (cit. on p. 2).

[71] X. Zhang et al. "Statistical Privacy for Streaming Traffic". In: *Proceedings 2019 Network and Distributed System Security Symposium* (2019). URL: https://api.semanticscholar.org/CorpusID:142503765 (cit. on pp. 2, 18).

[72] Y. Zhu et al. "Anonymity analysis of mix networks against flow-correlation attacks". In: *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.* Vol. 3. 2005, 5 pp.–. DOI: 10.1109/GLOCOM.2005.1577959 (cit. on pp. 2, 9).

[73] J. L. Zittrain et al. "The Shifting Landscape of Global Internet Censorship". In: Accessed: 2024-11-17. Berkman Klein Center for Internet & Society, 2017. URL: https://www.ssrn.com/abstract=2993485 (cit. on pp. 1, 3).

NOVA SCHOOL OF SCIENCE & TECHNOLOGY