TÉCNICO LISBOA

Enhancing the Tor Anonymity Network with K-Anonymous Flashmobs

José Eduardo Sapina Teixeira Brás

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor(s): Prof. Nuno Miguel Carvalho dos Santos
Prof. Diogo Miguel Barrinha Barradas

Examination Committee

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

November 2022

Dedicated to my mother.

Acknowledgments

First and foremost, I want to thank my advisors, Professor Nuno Santos and Professor Diogo Barradas for accepting me as a student. It has been a pleasure working with you both and owe you for the patience, guidance and the countless discussions that we had during the course of this year which have made this thesis possible. It was an amazing experience and I feel like I have truly learned a lot.

Second, I stand in recognition of the effort that Vítor Nunes and Kevin Gallagher have put into this project. Without you this thesis would have not been the same and I really appreciate the input and feedback that you both provided on our meetings.

I want to thank the friends I have made along my journey in Instituto Superior Técnico, namely, João Meira, Marco Coelho, Miguel Barros, Duarte Nascimento, and Diogo Martins. You guys not only helped me endure and persevere through these hard years in IST, but through your friendship you made me an overall better person.

I owe this journey in higher education to my family, above all, my mother Maria da Cruz and my father Manuel Isaac. I am eternally grateful for what you have always provided me and for your unconditional love, kindness, and support.

Finally, I wanna thank my girlfriend Iulia for always being with me and standing by my side. I will always owe you for your love, friendship and comprehending me like no other could. This one is for us too.

To each and every one of you – Thank you.

Resumo

Redes de anonimato como a rede Tor são ferramentas poderosas para aumentar a anonimidade e segurança das comunicações entre os seus utilizadores. A popularidade da rede Tor muitas vezes acaba por dar azo à sua utilização por parte de wistleblowers ou informantes, dispostos a revelar informações confidenciais ou atividades usualmente envolvendo organizações privadas, públicas ou governamentais. No entanto, devido à exposição de documentos confidenciais, estas pessoas podem ser acusadas de crime por entidades poderosas por terem atuado em prol da transparência e liberdade. Nesta tese propomos uma tecnologia com vista a aumentar a privacidade da rede Tor através de uma nova primitiva que permite proteger os seus utilizadores contra ataques iniciados por adversários globais passivos. Em particular, introduzimos a ideia de permitir utilizadores a criar uma *flashmob de k-anonimato*, i.e., um grupo que permite a cooperação de $k-1$ voluntários com intuito a proteger a liberdade de expressão se conectarem simultaneamente à Internet através do Tor e gerar tráfego dissimulado que irá permitir o organizar da flashmob desaparecer na multidão assim gerada. De forma a materializar esta ideia, propomos construir um novo *pluggable transport* para o Tor chamado Moby. Através da rede Moby, um adversário global passivo com a habilidade de intercetar todas as comunicações ao redor do globo não terão a capacidade de desanonimizar as comunicações dos participantes da flashmob mesmo utilizando o estado da arte das técnicas de correlação de tráfego.

Palavras-chave: Tor, K-Anonimato, Ataques de Interseção, Divulgação Estatística

Abstract

Anonymity networks such as Tor are powerful tools to increase the anonymity and security of user communications. The popularity of Tor leads to its usage by whistleblowers or informants, willing to reveal confidential information or activities often involving private, public or governmental organizations. However, by exposing classified material, these people may face prosecution from powerful actors by acting for the sake of liberty and freedom. In this work we propose a privacy-enhancing technology that provides a new primitive for securing Tor users against attacks mounted by global passive adversaries. In particular, we introduce the idea of allowing users to convene a *k-anonymous flashmob*, i.e., to leverage the cooperation of $k-1$ freedom-fighting volunteers to connect simultaneously to the Internet through Tor and generate covert traffic that will help the flashmob organizer blend into the crowd. To materialize this idea, we propose to build a new Tor pluggable transport named Moby. Using Moby, a global passive adversary with the ability to intercept all the communications across the globe will not be able to uniquely deanonymize the communications of flashmob participants even when using state-of-the-art traffic correlation techniques.

Keywords: Tor, K-Anonymity, Intersection Attacks, Statistical Disclosure

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis addresses the problem of intersection attacks and statistical disclosures on TorK and implements an extension allowing whistleblowers to protect their real identities against global-level adversaries performing passive long-term traffic analysis of network flows. Providing robustness against intersection attacks implies that an adversary that has a global view of the entire network (clients, relays and destination) should not, by any passive or active means, be able to find a link between clients and their destinations. However, due to several practical limitations (e.g. performance, reliability, maintainability), the problem of resisting intersection attacks and statistical disclosure in Tor has remained an elusive goal.

1.1 Motivation

Whistleblowers or informants are people who reveal privileged information or activities within a private, public, or government organization that are deemed illegal or fraudulent. To merely illustrate the core scenario of our work picture Alice, a notorious whistleblower. By openly denouncing wrongdoing through the publication of information on dedicated websites like WikiLeaks or Football Leaks, Alice might expose herself to prosecution or harassment. To protect her identity, she can resort to state-of-the-art anonymous communication networks such as Tor [1]. Tor's onion routing scheme [2] allows Alice to upload sensitive content on whistleblowing websites while hiding her true IP address [3].

However, over the years, the ever-present looming threat of deanonymization attacks launched at a large scale by global state-level adversaries has been greatly increasing [4]. Such attacks constitute a great risk to Tor users, like Alice, and emerge as a combination of four main different factors. Firstly, a growing consolidation of the Internet infrastructure in the hands of a few global players like Google and other large ISPs make it increasingly easier to collect large portions of Tor traffic from few vantage points [5]. Second, Tor guard nodes – which act as entry-points to the Tor network – are skewed toward a relatively small number of ISPs, making it even more practical to probe into the Tor network and intercept Tor users' communications [6]. Third, the sophistication of traffic analysis techniques is developing at a fast pace due to the application of machine learning algorithms that concur to the building of accurate

traffic classifiers that can be used for launching correlation attacks [7, 8]. Finally, with the advent of mass surveillance efforts partially enabled and justified by anti-terrorism or the pandemic, states might use their law enforcement agencies in upholding mandates to monitor anonymous communication networks, therefore being able to deanonymize potential whistleblowers [9–11].

Over the last few years, there have been many proposals of anonymous communication systems [12–17] that allow Alice to go incognito and hide her identity from the aforementioned major players. Although providing strong anonymity properties, these systems are often vulnerable to correlation attacks [18]. Due to Tor being the most widely used anonymous communication network, recent works have proposed systems to mitigate this family of attacks against onion routing, such as TorK [19], a pluggable transport leveraging the notion of indistinguishable flows to ensure k-anonymity amongst multiple sources of Tor connections. Still, resisting a global passive adversary – one that can see and correlate all traffic in the network, but cannot modify or interact with it – able to perform intersection attacks and statistical disclosure over a long period of time remains an open research challenge [20, 21]. Although there have been some proposals to fix these everlasting issues [22, 23], none have been prototyped over onion routing-based anonymity schemes, and only a few anonymity networks claim sturdiness against such attacks [24–31].

In order to understand the prevalence of these attacks on Tor, imagine that Alice wants to publish information about the violation of civil liberties by her employer, a state-level security agency. Alice can connect to WikiLeaks using TorK and never reveal her personal information when exposing incriminating data. However Mallory, the director of the security agency Alice is employed in, can order network administrators to monitor WikiLeaks and Alice's posting patterns over a long period of time. By observing the user churn in the variable k-anonymity sets and performing the intersection of these, Mallory is able to narrow sets down to one user, Alice. Furthermore, if Mallory focus her efforts in analysing the accesses to WikiLeaks over this long-term passive analysis she will also be able to determine with some probability p the chances for Alice to be the poster of incriminating data against her agency.

To mitigate these risks, we propose Moby, a new privacy-enhancing tool that provides resistance to the pitfalls of TorK [19] such as client churn. Moby will also be the first available tool able to provide resistance against intersection attacks and statistical disclosure on the Tor network. These issues are solved through an abstraction that we have properly named $flashmobs$, building on the concept of a spontaneous group of individuals bound by time performing a specific set of joint actions. By enforcing multiple users to perform web browsing actions on a scheduled time period we allow whistleblowers to go incognito, blending in with a group of individuals with a predetermined communication pattern. If these flashmobs are scheduled to be performed in rounds with the same fixed set of k users accessing the same websites, no global passive adversary can perform an intersection or statistical disclosure attack against our system. The reason for this being that the intersection of flashmob user sets over multiple rounds is the same as the original membership set since churn is not tolerated. Furthermore, through this condition an attacker is not able to perform a statistical disclosure attack to distinguish the traffic of each user from another.

With this being said we have extensively evaluated Moby. Using Raspberry Pi devices, we deployed

Moby hubs in nine residential networks and measured the performance of our system while routinely composing k-circuit flashmobs (with $k = 9$), with each round happening in 2h intervals over the course of one week. In all this time, all the participating hubs were available, enabling users to access the Tor network with performances varying between 1Mbps and 3Mbps. Finally, in order to resist against a statistical disclosure attack, we studied several different configurations, to determine how Moby is able to perform flashmobs that give users strong anonymity and plausible deniability guarantees.

1.2 Contributions

This thesis studies methods to make Tor sturdier against a global passive adversary, specifically, strengthen TorK against intersection attacks and statistical disclosure. It presents Moby, an Add-On for TorK responsible for leveraging a concept named flashmobs with the intent of coordinating TorK users on anonymity sets to shield one another against this category of attacks and makes the following contributions:

- Design of Moby including four components: (i) concept of flashmobs to resistic intersection attacks, (ii) how to recruit clients and deal with churn, (iii) design of mesh of Moby bridges to balance workloads, and (iv) how to deal with the traffic of flashmobs – mostly how to fake realistic browsing patterns against a global adversary.
- Implementation of the Flashmob concept, allowing for creation, scheduling and performance of traffic generation during a set amount of time for a predefined number of rounds.
- Integration of Moby with the previous anonymization mechanisms implemented on the TorK system, mostly the aspects related with traffic shaping and the creation of *k*-circuits.
- Deployment of Moby in Internet of Things (IoT) devices scattered behind residential networks – presenting a high availability real world scenario.
- Evaluation of Moby for performance metrics, both using a virtual test bench on Docker and actual deployed microcomputers.
- Evaluation of Moby for statistical disclosure resistance against a global passive adversary.

Having stated our contributions, some of them, specifically, (iii) the design of a mesh of Moby bridges, and (iv) faking realistic browsing scenarios, ended up not being implemented due to the shortage of time and the fact that many parameters of our goals having to be readjusted due to challenges presented on the deployment and evaluation of Moby.

1.3 Thesis Outline

The rest of the report is organized as follows. Section 2 introduces the required background. Next, we discuss the related work in Section 3. In Section 4 we describe the design and architecture of Moby.

In Section 5 we delve into the implementation of our system and its features. Section 6 describes how we evaluated Moby and the following results. Finally, Section 7 presents the schedule of future work and Section 7 concludes the report.

Chapter 2

Background

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

2.1 The Tor Ecosystem

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

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

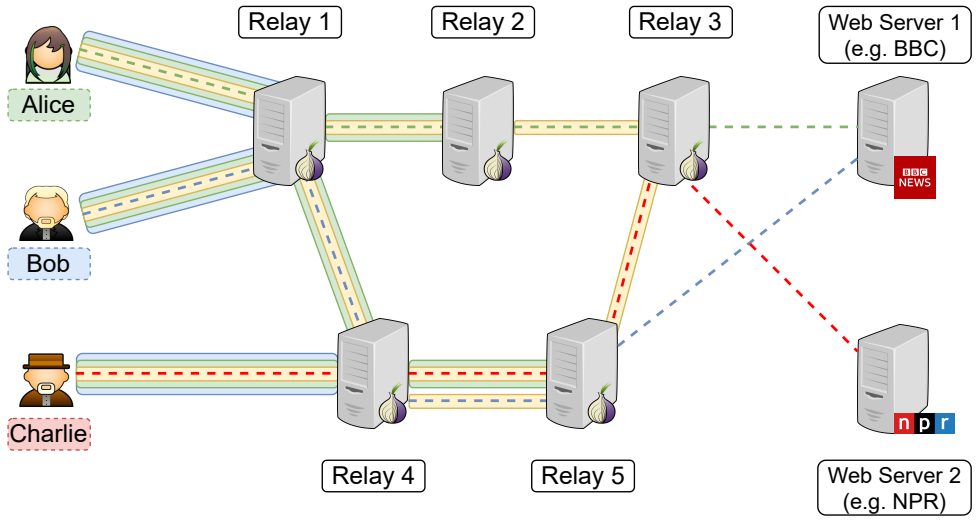


Figure 2.1: Examples of vanilla Tor circuits. Alice constructs a circuit $(1 \leftrightarrow 2 \leftrightarrow 3)$ to access BBC. Bob constructs a circuit $(1 \leftrightarrow 4 \leftrightarrow 5)$ in order to also access BBC. Charlie constructs a circuit $(4 \leftrightarrow 5 \leftrightarrow 3)$ in order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

2.1.1 Bridges

Given that Tor is very popular as an Internet circumvention tool within countries ruled by repressive regimes, Tor has become a major target of blocking by the ISPs of such countries. One way for blocking Tor traffic is by blacklisting Tor relays. Since the relay list is public it is trivial to blacklist the IP addresses of all Tor relays, preventing Tor clients from establishing circuits. To circumvent this attack, Tor employs *bridges*. Bridges consist of unlisted proxies whose goal is to forward client's traffic to an entry relay. Thus, rather than connecting to some potentially blacklisted entry relay, clients connect instead to some unlisted bridge (hopefully) unknown by the adversary. Some bridges are *public*, others *private*. Public bridges are deployed by volunteers to be used by any Tor user. Private bridges are exclusively for those who know about their existence [32].

However, bridges are still vulnerable to traffic fingerprinting attacks. If a user connects to a bridge using the vanilla Tor protocol, it becomes possible to identify Tor traffic and blacklist that bridge. Since Tor uses fixed-size 512 byte cells, the resulting packet size distribution exhibits distinguishable patterns that can help a threat actor to identify with high probability if a given flow carries Tor cells or not. Tor traffic can be detected also based on TLS byte patterns. Tor communications (e.g., between relays) are encrypted using TLS. Server Name Indication (SNI) is a TLS extension [33] that adds the domain name to the TLS header in order to be used in the *Client Hello* handshake phase. However, the SNI is not encrypted – since it occurs before the TLS handshake – allowing an eavesdropper to discover the domain name that the client is trying to reach. Tor uses a fixed random string as a bogus domain name (e.g. www.kf3iammnyp.com) and a specific cipher suite set [34]. These properties uniquely identify encrypted traffic as Tor's, enabling censors to block new bridges minutes after they have been deployed by simply observing these two properties.

2.1.2 Pluggable transports

To surpass identification, Tor bridges support *pluggable transports* (PT) [35] which are obfuscation wrappers that shield the Tor traffic between a client and a bridge from traffic analysis. The most popular PTs are *meek* [36] and obfs4 [37], which is based on obfs2 [38] and provides a level of obfuscation for an existing authenticated protocol, like SSH or TLS. The obfs4 protocol encrypts all traffic using a symmetric key shared between the bridge and the user derived from both client's and bridge's initial key [34]. An attacker would see a randomly encrypted byte stream. *meek* uses a technique called *Domain Fronting* [36]. This technique consists on using different domain names in the SNI and in the HTTP *Host*. The idea is to encode a legitimate website domain name into the SNI (e.g. the address of a public cloud) and use the Host field in the encrypted HTTP request to ask for access to the forbidden one. *meek* requires a CDN (Content Delivery Network) like public cloud providers which supports domain fronting. Bridges can support multiple PTs in order to serve a larger number of clients. However, bridges that offer more than one PT may reduce the security of the most secure PTs they offer. Similarly, bridges running non-Tor services (e.g. SSH) also reduce the level of security provided to users [32].

Snowflake is a recent pluggable transport which is composed of two major components: (i) volunteers running Snowflake proxies, (ii) a broker delivering snowflake proxies to users interested in circumventing censorship. Snowflake uses domain fronting to create a connection between a snowflake proxy run by a volunteer and a censored Tor user. These proxies are lightweight, ephemeral, and easy to run, allowing to scale Snowflake in the presence of thousands of users. For censored users, whenever a Snowflake proxy gets blocked, a broker will find a new proxy for this user automatically.

2.2 Attacks Leveraging Malicious Tor Entities

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

2.2.1 Timing attacks

These are end-to-end active attacks where an attacker attempts to determine the endpoints of a circuit by correlating the time it takes for a flow to travel from the entry to the exit relay. Chakravarty et al. [39] employ single-end bandwidth estimation for deanonymizing the source of a given Tor connection. Pries et al. [40] presented a different timing attack based on the disruption of the AES-CTR counter used to encrypt Tor cells. To mitigate timing attacks, Murdoch et al. [41] suggests some defensive strategies. However, these techniques cannot be adopted in Tor as they would increase the communication latency

beyond what can be tolerated for a low-latency anonymity network. Instead, Tor relays send cells from different streams in a round-robin fashion.

2.2.2 Sybil attacks

The attacker deploys malicious relays in the Tor network, while creating the illusion of pertaining to different entities. The idea is to obtain a large penetration in the network [42] giving the attacker many vantage points to intercept user traffic. This level of control bolsters the attacker's ability to launch correlation or timing attacks [43]. Defenses against Sybil attacks are not trivial. One approach is to relay on a central authority to endorse the relays that can join the network [44]. However, this introduces a single point of control which defeats the very purpose of the Tor network. Another approach [45] is to limit the number of new accepted relays by IP address subnet, forcing the attacker to control multiple subnets in order to mount the attack. Both solutions do not directly stop sybil attacks but increase the attacker's startup cost.

2.2.3 Predecessor attacks

Repeated communication between two endpoints of a Tor circuit may open the door to vulnerabilities that can be exploited by a traffic analysis-capable adversary [46]. In order to perform a predecessor attack, an attacker must compromise an entry and an exit relay. The goal of this attack is to learn the identity of a single or multiple senders when they are connected to a destination over time. Wright et al. [47] present a defense against predecessor attacks assuming a static network model, i.e. nodes do not leave the network. To defend against predecessor attacks, the authors propose to fix relays of onion circuits in certain positions (e.g. entry, or exit). Tor uses the same approach by imposing guard rotation restrictions [48], as further discussed in Section 3.1.

2.3 Attacks Leveraging Traffic Analysis

Correlation attacks *per se* are passive attacks which only require an adversary to be placed in a privileged position in the network and to observe incoming and outgoing network flows at Tor connections' endpoints. Albeit such attacks do not require the compromise of nodes nor the perpetration of active attacks, such attacks can provide an adversary with an advantage to perform correlation.

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

2.3.1 Circuit fingerprinting attack

Sun et al. [49] present an approach based on asymmetric traffic analysis allowing an adversary to correlate and deanonymize a circuit's endpoints, even if the attacker has access to different directions of

the flow, e.g., from client to entry and server to exit. The rationale of this technique is based on the fact that the Internet relies on asymmetric connections, i.e. the path from the client to the server may differ from the same server back to the client, and that an adversary is still able to perform traffic correlation by observing different directions of a given flow. To augment the possibility of an adversary to observe a flow, authors additionally propose a BGP interception attack which can be launched by a malicious AS in order to divert traffic, enable traffic analysis, and forward traffic to the original destination.

2.3.2 Correlation-based analysis

One of the first correlation attacks based on timing analysis was proposed by Shmatikov and Wang [50]. Inter-packet timing information is usually not carefully protected in mix networks since it would require to delay packets to hide timing patterns. An attacker can exploit this timing property by correlating the inter-packet time on both endpoint links, concluding that those links belong to the same circuit, which would tie a source to the corresponding destination.

One of the machine learning algorithms used to conduct correlation attacks for Tor is DeepCorr [51]. DeepCorr uses advanced machine learning algorithms to conduct accurate flow correlation on Tor. DeepCorr learns a correlation function which is able to link flow samples regardless of their destination, while accounting for the unpredictability of the Tor network. To protect against traffic correlation attacks based on machine learning techniques, as in DeepCorr, Nasr et al. [51] propose that Tor should enforce the use of pluggable transports across all relays, instead of just on bridges as in vanilla Tor. However, this solution would incur in significant performance degradation. Alternative countermeasures rely on employing AS-aware relay selection mechanisms [52–54] that effectively decrease the probability of an adversary to be in the necessary position to observe traffic and perform a correlation attack.

As of the year of writing, DeepCoFFEA [55] is the state-of-the-art algorithm of traffic correlation on Tor. It relies on end-to-end correlation of Tor flows using Deep Learning and an amplification technique which divides flows into short windows and uses voting across these windows to significantly reduce false positives. With the usage of windows, embedding networks can be used independently lower the false positive rate, thus achievement great deanonymization results against Tor.

2.4 Attacks Leveraging Anonymity Sets

The next category of attacks takes a different angle and explores Tor's limitations at maintaining large anonymity sets for individual users. If at a given point in time, a passive adversary can enumerate the IPs of every user accessing the Tor network (say, k users in total) and of all destinations being accessed from every Tor relay, then the attacker can guess, with a probability of at most $1/k$, that a given user has visited a particular destination. The higher the size k of a user's anonymity set $\mathbb{K}$, the stronger the anonymity will be. However, it is possible to attack Tor (and other existing anonymity systems) as a result of a continuous erosion of users' anonymity sets. Next, we cover two main attacks:

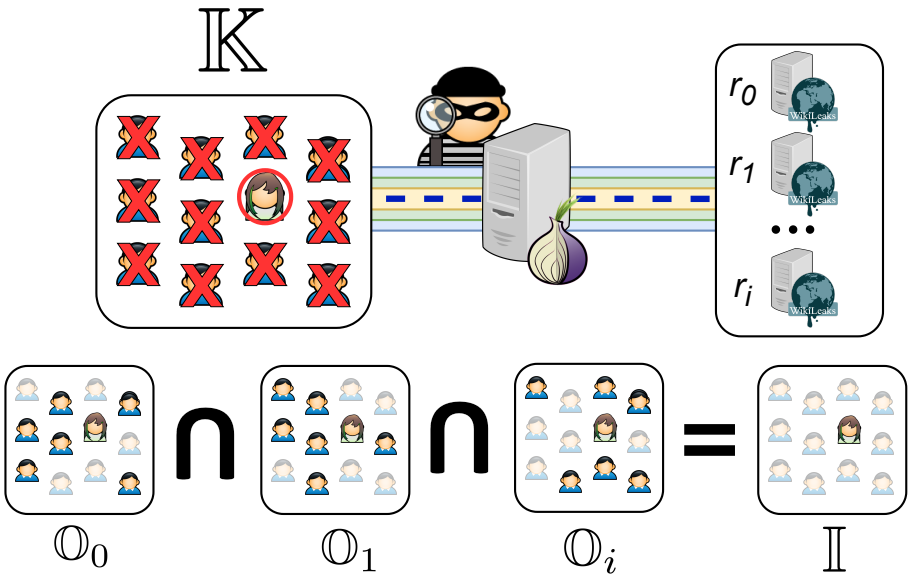


Figure 2.2: Example of an intersection attack to anonymity set $\mathbb{K}$. The attacker observes the online user sets $\mathbb{O}_i$ and the traffic to WikiLeaks over multiple rounds r_i. The intersection of all sets $\mathbb{I}$ allows to infer the real identity of Alice.

2.4.1 Intersection attacks

Throughout the passive observation of communications, an attacker can determine which users are online and the destination of their messages. Since typically the pool of destinations (e.g., websites) one accesses when online is limited, through repeated observations an attacker can start differentiating the traffic pattern of each user. For instance, a typical user tends to visit the same destinations over different sessions. Intersection attacks leverage such patterns by intersecting different sets of users active at any given moment in order to gain information and differentiate what websites a user is accessing. Although modern anonymous communications networks offer security against powerful adversaries, most systems of this type falter against a global passive adversary capable of pervasive network traffic analysis in the presence of churn. With numerous users logging in and out and through the sending of linkable messages users can quickly lose their anonymity. Intersection attacks are a well known open problem and very difficult to solve in an efficient manner [4, 20].

To give a better intuition of how intersection attacks work, consider Figure 2.2. Let us assume Alice wants to post meaningful information about the corruption in the national security firm she is working for on the WikiLeaks website using an anonymity system. Alice will place her posts over multiple rounds r_0 to r_i. In each iteration, a global passive adversary takes note of the website accesses and its respective online user anonymity sets $\mathbb{O}_0$ to $\mathbb{O}_i$. The group of online clients varies in each iteration due to user churn. Assuming the adversary monitors all accesses to WikiLeaks over a long period of time, the intersection of all user sets $\mathbb{I}$, allows for the singling out of Alice progressively reducing the anonymity set size k to 1, i.e., Alice herself.

Most of the works providing intersection attack defenses agree on the usage of a fixed size anonymity set whilst sensitive website are accessed. Wolinsky *et. al.* [22] provide the seminal work on intersection attack defenses. The said defense consists in the creation of a fixed anonymity set to perform round-

based posts on specific websites. A similar approach is also used by Hayes *et. al.* [23], the main difference residing on the threshold used to contain user churn. Whilst the first uses an oracle to control the threshold related to the user churn in order to prevent an intersection attack, the latter does not set such threshold allowing for users to be deanonymized more easily. Despite this flaw, the authors argue that whilst not providing strong anonymity their system provides less latency. One of our concerns is that both of these approaches, in the presence of churn, after a few iterations require users to change their identifiers. Such switch of user identifiers effectively deprecates the older instance prohibiting further website accesses with the same identifier. We elaborate on both of these works further down in the next section analysing their algorithms and system architecture.

2.4.2 Statistical disclosure attacks

As explained above, intersection attacks work in a deterministic fashion allowing for the intersection of different user anonymity sets to link a client to a destination. A statistical disclosure attack, on the other hand, is constructed probabilistically. Assuming that, after conducting multiple observations, an attacker focuses on a specific destination, then it is possible to estimate the probability of those messages pertaining to a specific sender using different strategies. Oya et al. [21] propose a compelling analysis regarding the uniformization and comparison of the different categories of statistical disclosure attacks. We base our mathematical analysis on the aforementioned work for easier reader comprehension and will analyse three main statistical disclosure variants: (i) the original statistical disclosure attack, (ii) the generalized statistical disclosure attack and (iii) the least squares statistical disclosure attack.

To present the mathematical models employed in each variant, we shortly review the notation used by Oya *et. al.* relevant for our problem. Firstly, let j be a destination, i the sender, and b background users. $p_{j,i}$ and $p_{j,b}$ represent the probabilities of traffic reaching j being from sender i and background users b respectively. Messages sent and received respectively by users i and j in round r are represented as u_i^r and y_j^r. Furthermore, lowercase letters represent vectors. Let the superscript T represent the algebraic transposing operation, the column vector containing all messages sent by user i from round 1 up to round ρ is represented by $\boldsymbol{u}_i = [u_i^1, u_i^2, ..., u_i^\rho]^T$. Reciprocally, the number of messages received by destination j is defined as $\boldsymbol{y}_j = [y_j^1, y_j^2, ..., y_j^\rho]^T$. Finally, the tilde mark "˜" placed on top of $\tilde{\boldsymbol{u}}_i^r$ denotes a binary representation of $\boldsymbol{u}_i^r$ disclosing whether there is at least one message sent by user i in round r ($\tilde{\boldsymbol{u}}_i^r = 1$) or not ($\tilde{\boldsymbol{u}}_i^r = 0$). The vector notation of the aforementioned symbol is $\tilde{\boldsymbol{u}}_i = [\tilde{u}_i^1, \tilde{u}_i^2, ..., \tilde{u}_i^\rho]^T$

1. *Original statistical disclosure attack*: Having analysed the required notation to understand the probabilistic aspect of the attack, let us now analyse the original statistical disclosure proposed by Danezis [56]. The author makes the assumption in his original paper that sender i does not send more than one message in each communication round and other background traffic reaching j is uniform, i.e. $p_{j,b} = \frac{1}{N}$ for $j = 1, 2, ..., N$. The attack is thus modeled as:

$$\tilde{\boldsymbol{u}}_i^T \boldsymbol{y}_j \approx \tilde{\boldsymbol{u}}_i^T \mathbf{1}_\rho \cdot p_{j,i} + \tilde{\boldsymbol{u}}_i^T (\mathbf{1}_\rho \cdot t - \mathbf{1}_\rho) \cdot p_{j,b} \tag{2.1}$$

2. *Generalized statistical disclosure attack:* The original statistical disclosure attack assumed that the sender i would only send a message per communication round. Matthewson and Dingledine [57] extend on the sender behavior and assume a scenario where i might send j more than one message. This generalized statistical disclosure is more practical and closer to real world applications. On their work the authors theorize their attack against mix networks and other anonymous networks [58]. The generalized statistical disclosure attack is defined as in the equation below. This equation does not include the number of messages sent by user i each round – 1_ρ. Instead, it uses the number of actual messages sent by i, u_i, with $1_\rho \cdot t - u_i = u_b$.

$$\tilde{\boldsymbol{u}}_i^T \boldsymbol{y}_i \approx \tilde{\boldsymbol{u}}_i^T \boldsymbol{u}_i \cdot p_{j,i} + \tilde{\boldsymbol{u}}_i^T \boldsymbol{u}_b \cdot p_{j,b} \tag{2.2}$$

3. *The least squares statistical disclosure attack:* Finally, Pérez-Gonzalez [59] proposed a profiling attack based on the maximum likelihood of estimating user profiles by solving the Least Squares problem. Least Squares statistical disclosure ensures that the mean squared error between the real and estimated user profiles is minimized. Let $\hat{p}_{j,k}$ represent the estimator from all outputs when estimating p_{j_k}, the equation of this attack is given as:

$$\boldsymbol{u}_i^T \boldsymbol{y}_j = \boldsymbol{u}_i^T \sum_{k=1}^{N} (\boldsymbol{u}_k \cdot \hat{p}_{j,k}) \ , \ for \ i = 1, ..., N \tag{2.3}$$

In summary, intersection and statistical disclosure attacks present a practical issue in existing anonymity networks such as Tor, since a global passive adversary may be able to link the messages received by a destination from a sender. As such, this makes it possible to debunk plausible anonymity of whistleblowers when accessing sensitive websites or posting leaked data.

Summary

This chapter introduces the notion of the Tor anonymity network and their main entities. As we can observe there are three main types of vulnerabilities that can be used against Tor, mainly: (i) attacks leveraging malicious Tor entities, (ii) attacks leveraging traffic analysis, (iii) attacks leveraging the composition of anonymity sets. Intersection and statistical disclosure attacks are implemented over the course of repeated observations of anonymity sets, singling out a user possibly accessing sensitive content, and thus defeating the concept of plausible deniability.

Chapter 3

Related Work

In this chapter, we introduce the main known defense strategies against traffic correlation attacks, intersection attacks, and statistical disclosure. We also discuss their strengths and limitations when facing a state-level adversary. Then we provide information on novel systems implemented to thwart these attacks. We discuss how the techniques used in these systems were never implemented and how to apply them for Tor.

3.1 Avoiding Unsafe Relay Nodes

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

3.1.1 Run a co-located trusted relay node

Clients may run a co-located trusted relay node alongside the Tor client, and use that relay as entry node to the Tor circuits created by the user. As a result, the downstream relay nodes will not be able to determine whether the traffic forwarded by the trusted relay node was originally produced by the local user or by another user that may be using that same relay node for building its own circuits. However, a significant number of clients are located behind restrictive firewalls where they cannot relay traffic [60].

3.1.2 Scan and flag bad relay nodes

A second approach, currently used by Tor, is to scan and identify bad relay nodes to decrease the possibility of clients selecting malicious relays, by detecting and reporting bad relay nodes. Generally, a relay is considered to be bad if it is malicious, misconfigured, or unreliable. To mark relays, Tor uses a set of labels designated as *flags* [61]. Tor maintainers run a service aimed at verifying the reports of possibly unsafe relays [62] and also bad exit nodes, using a tool named *exitmap* [63]. Additional tools

can be used for that purpose by the community in general, such as *torscanner* [1], and *tortunnel* [2]. Some of these tools use decoy traffic in order to detect bad relays [64].

3.1.3 Restrict the set of entry nodes used by a client

Lastly, Tor also restricts the set of entry nodes used by a client in an attempt to mitigate the guard rotation weakness [48, 65]. If a client was unlucky and selected a malicious guard, he had the chance of regaining anonymity when its current guard changed. However, such parameters were not strong enough to hold a large AS-level adversary; this is described as guard rotation weakness [48]. Elahi et al. [65] demonstrated that Tor's time based guard rotation criteria led to clients switching guard relays more often then they should, increasing the possibility of profiling attacks.

3.2 Avoiding Unsafe Autonomous Systems

Unfortunately, the techniques presented above may not be effective against an adversary that can observe large fractions of the network. Such an adversary may not even need to control any specific relay node to deanonymize Tor traffic, but only to possess the ability to eavesdrop on the inter-relay traffic that crosses the network controlled by the adversary [8, 66]. To cope with this problem, several authors have proposed new defensive approaches against AS-level adversaries, i.e., those with the ability to access the network infrastructure of an entire Autonomous System (AS). Typically, such approaches attempt to help Tor clients' to choose paths away from the prying eyes of malicious ASes by leveraging the analysis of the Internet topology boundaries and inter-relay latencies [67].

3.2.1 AS-level monitoring and tuning

As a way to prevent attacks based on traffic interception through BGP hijacking (see Section 2.3), Sun et al. [49] have proposed a monitoring framework for detecting BGP changes. The use of such framework allows Tor to inform vulnerable clients, which in turn can opt to suspend Tor communications or use another relay. In order to have a robust solution it is necessary to accurately know which ASes are traversed by a given circuit path. To this end, the proposed framework computes the traceroute of every Tor relay daily. Based on these results, it is possible to observe AS-level path changes and detect suspicious ASes.

3.2.2 AS-aware path-prediction

Several authors have proposed AS-aware path selection algorithms for decreasing the chance of an AS-level attacker to observe traffic flowing between Tor circuits' endpoints [49, 52–54]. Edman and Syverson [54] suggest adding two new requirements to Tor's path selection algorithm: i) mandates that each node in a circuit must be located in a different country, and ii) instead of requiring unique countries,

[1] https://code.google.com/archive/p/torscanner/ Accessed: 2022-01-05
[2] https://moxie.org/software/tortunnel/ Accessed: 2022-01-05

each node should be located in a different AS. While the two approaches decreased the probability of an AS to eavesdrop in both ends of a connection, they did not sufficiently mitigate the possibility for a malicious AS to perform traffic correlation. Akhoondi et al. [53] propose LASTor, an AS-aware Tor client that selects safe paths between a client and a destination. Sun et al. [49] that each relay publishes, as part of the Tor consensus document, the list of ASes it uses to reach a given relay or destination. Clients can then use this information along their own measurements when constructing circuits to select relays in a way that one AS will not appear on both the entry and the exit relay. Nithyanand et al. [52] further developed an AS-aware variant of Tor, called Astoria, which avoids vulnerable circuits while employing an efficient network management by load balancing circuits across secure paths. However, due to the existence of active BGP interception attacks [49], Astoria's Internet topology maps may become outdated for short periods of time, allowing an attacker to still launch a successful correlation attack.

3.3 Avoiding Unsafe Geographical Regions

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

Tor allows users to select a set of countries to exclude from circuit selection [60] i.e. regions to where it should not forward client's traffic. DeTor [68] presents techniques to prove that a Tor circuit did not travel within excluded regions. To provide the so called *provable geographic locations*, DeTor authors borrow the idea of *alibi routing* [69] into Tor. Alibi routing (AR) uses the packets' round trip time (RTT) and the speed of the light as a constant to prove that a given packet did not travel within forbidden regions. AR uses a single relay located outside the forbidden region to confirm that traffic is going from that relay to the destination. While AR uses one single relay, DeTor [68] generalizes this approach for three relays. However, there are a few limitations regarding DeTor which make it an unconvincing solution for providing provable geographical avoidance. First, DeTor obtains the list of relays from the Tor's public database which contains several bits of information, such as: IP address, port, public key and country. DeTor may also use IP geolocation services to find the exact location of Tor relays, but without any further confirmation. This may hamper the accuracy of DeTor's measurements [70, 71]. A second limitation involves handling links with high latency, where it is not possible to measure the packets' travel times accurately solely relying on RTT measurements. Another limitation is that DeTor assumes a symmetric routing nature, i.e. assumes that the request and reply will traverse the same geographical locations, something that may be unlikely to happen in practice.

A more recent piece of work by Kohls et al. [71] introduces the concept of *empirical avoidance* and proposes new improvements to overcome DeTor's main limitations. In their work, authors propose TrilateraTor, a system introducing a new measurement technique that derives a circuit end-to-end timing directly from the handshake in Tor's circuit establishment procedure. To prevent the use of fake GeoIP information in its measurements, TrilateraTor leverages a distributed measurement infrastructure so as to perform trilateration and obtain accurate estimates of the physical location of Tor nodes.

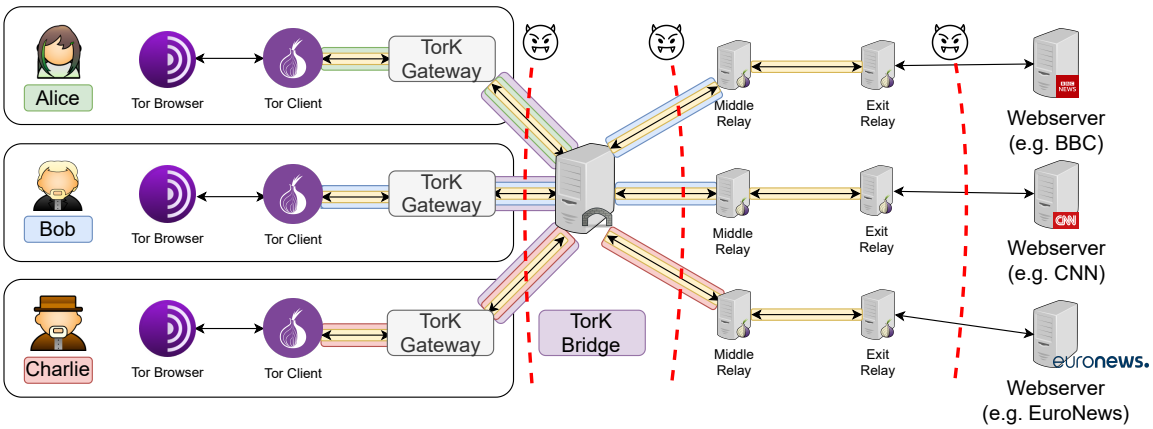


Figure 3.1: TorK architecture representing a k-circuit. All hops are observed by the attacker – dashed red line. All the three members opened a Tor circuit.

3.4 Avoiding Global-Level Traffic Correlation Attacks

One of the main vulnerabilities of Tor is correlation-based traffic analysis (see Section 2.2). Nunes [19] proposed TorK with the goal of providing a defensive measure against these attacks under a very strong threat model, where the threat actor is assumed to impersonate a global passive adversary capable of eavesdropping on the entire Tor network. Put simply, the central idea is to increase the anonymity set associated with the source IP address of a given Tor circuit from one single user to k plausible users. Figure 3.1 depicts the architecture of TorK which includes two components: a TorK client and TorK bridge. Client and bridge implement a TorK-specific pluggable transport. As a result, TorK is fully compatible with the existing Tor components and client-side applications.

Figure 3.1 also shows how Alice leverages TorK to communicate with a given web server through a standard Tor circuit. As usual, this circuit is made up by three relays – entry, middle, and exit – and the access from the client to the entry node is mediated by a bridge (the TorK bridge). Assuming the existence of a global-level adversary that can observe all network packets exchanged in this communication, using simply a standard vanilla Tor circuit, the adversary would be able to deanonymize Alice's client by launching a flow correlation attack.

TorK prevents this attack by allowing $k - 1$ additional users (in the figure, just Bob and Charlie) to collaborate with Alice so that the adversary will not be able to distinguish who amongst them is the real originator of traffic associated with this circuit. To achieve this, prior to the circuit establishment phase, all the k users use the local client to connect to the bridge. We call each of these channels a *segment*. Each segment acts as a tunnel between the local client and the bridge such that the local client can send arbitrary traffic through it. However, while Alice's segment will be used to transmit real data – i.e., the packets of the Tor circuit – the segments of the supporting users will transmit chaffing payload. The bridge discards the chaff and only lets Alice's actual traffic to be forwarded to the entry node and delivered to the intended destination.

To prevent the adversary from distinguishing which segment carries the Tor circuit's cells (this could be achieved by observing differences in the volume and timing properties of the traffic), the traffic of

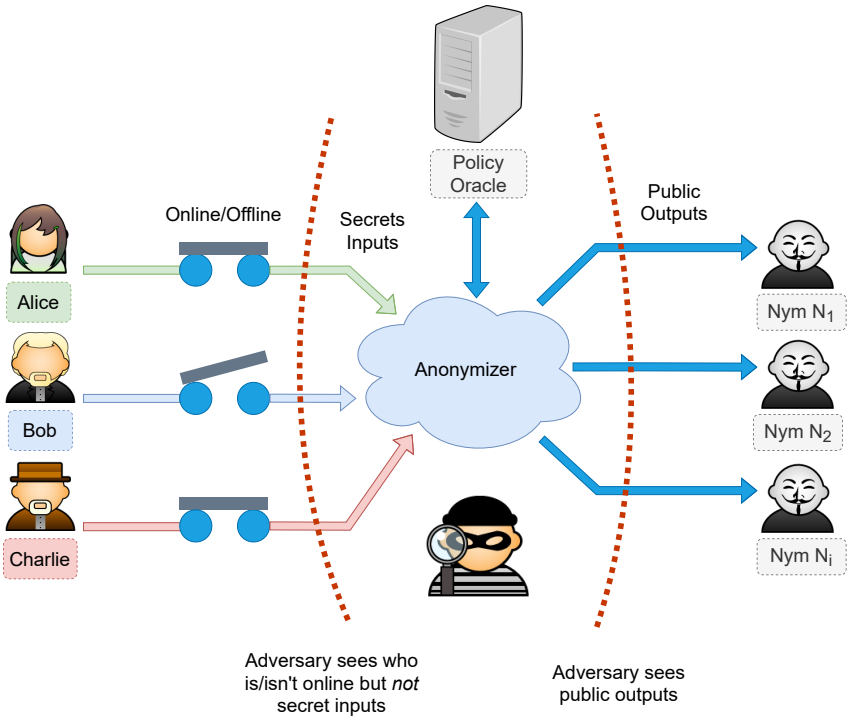


Figure 3.2: Conceptual Design of Buddies. The adversary observes both the secret inputs and outputs but cannot determine whether users are online or offline. The public output might be either (i) an actual message or (ii) a null message.

all participating segments is encrypted and modulated according to a common traffic shaping function. Thus, even though the attacker is able to capture and inspect the traffic from each of the three users it cannot correlate the message to the original sender, i.e. from the three users discover the one that is transmitting the messages due to indistinguishability between segments. The attacker can inspect all hops in the network but he would have to randomly guess the sender having a $1/k$ probability of succeeding. As a result, TorK provides k-anonymity by allocating groups of size k and introduces the term *k-circuit* to refer to a coordinated setup of k segments for tunneling Tor circuits.

3.5 Avoiding Weaknesses for Anonymity Sets

Despite the anonymity improvements brought about to Tor by TorK, which can thwart traffic analysis attacks whilst being able to preserve k-anonymity, TorK is still vulnerable to intersection and statistical disclosure attacks.

With the possibility of clients connecting in and out of bridges, a global passive adversary is able to keep track of the k-anonymity set and the websites being accessed correlating clients with their destinations. The churn present in TorK, especially the fact that the system allows for recurrent connections facilitates the intersection of i sets throughout a period of time effectively deanonymizing clients and weakening its k-anonymity property. This section delves into general solutions that aim at shielding anonymity sets from global passive adversaries.

Wolinsky *et. al.* proposed Buddies[22], the seminal work about intersection attack resistance. This

system allows for strong anonymity properties [72] bulking anonymity sets in the face of a global passive adversary. Figure 3.2 describes the proposed conceptual model of the Buddies system. Let us assume Alice wants to access the WikiLeaks website and post information about her corrupt employer without having her identity disclosed. Buddies will group Alice with a fixed set $\mathbb{K}$ of other $k-1$ users with each user accessing a predetermined sensitive content website in round robin fashion. Instead of using her unique identifier, Alice will rely on a pseudonym dubbed *Nym* [73] essentially working as an anonymous online handle. Buddies will attribute to Alice Nym N_1 when she joins the system and further WikiLeaks posts will be under her new handle instead of her real identity. Everytime Alice wants to post information on WikiLeaks with her Nym N_1 the users on her anonymity set – Bob and Charlie – will generate cover traffic to be sent to the Anonymizer. The Anonymizer is simply a black box anonymous communication network in the Buddies conceptual model focused on the secure and untraceable routing of traffic – the authors based their prototype on Dissent [13, 74, 75]. However, before allowing the message to be sent, the Anonymizer must first check whether it is still safe to use the same Nym. To this end, it queries the Policy Oracle which computes how much group-intersection information would be leaked allowing an adversary to mount an attack. If this information reaches a certain threshold, then that Nym is no longer considered safe and the user is informed to adopt another Nym.

More specifically, the Policy Oracle works as follows. Each round Alice wants to access Nym N_1, the encrypted messages sent from the subset of online users $\mathbb{O}$ will be routed from the Anonymizer to the oracle. The other users in $\mathbb{O}$ generate cover traffic when not accessing their attributed Nym. The Policy Oracle mimics an intersection attack based on the current subset $\mathbb{O}$, Alice is included in, and the several online users $\mathbb{O}$ subset data gathered over multiple rounds. Buddies allows for the configuration of a threshold essentially dictating whether the size of $\mathbb{O}$ shields Alice when posting to N_1. If N_1 has been compromised, under the current threshold values, Alice is forced to abandon N_1 and switch to a new Nym for further accesses. The Policy Oracle thus acts as a filter, removing Alice from subset $\mathbb{O}$ if she risks having her identity disclosed. The Policy Oracle filtered subset is represented by $\mathbb{P}$. As such, if no filtering was done $\mathbb{P} = \mathbb{O}$, whilst if Alice was removed from the subset we have $\mathbb{P} \subseteq \mathbb{O}$. If Alice is removed from the subset, when trying to post under N_1, instead of her post being routed to the destination, one of the cover messages from $\mathbb{P}$ is sent instead. This essentially obfuscates an adversary vision in two ways: (i) with cover traffic being sent the adversary cannot tell whom N_1 owner (Alice) actually is and (ii) it is not obvious whether Alice was online but filtered or actually offline. Despite having garnered a significant attention from the research community, Buddies remains to be prototyped and evaluated on onion routing-based anonymity networks, most notably Tor.

Claiming that Buddies induces excessive communication latency and may prevent users from posting messages, Hayes *et al.* developed TASP [23], a system with the goal of protecting anonymity networks from intersection attacks while providing strong anonymity and lower latency. Similar to Buddies, TASP preserves fixed anonymity sets. Differently from Buddies, however, it groups users into anonymity sets based on their traffic patterns allowing the generation of more effective cover traffic. To group clients into such sets it uses machine learning-based traffic analysis algorithms working in two phases of operation, a learning phase and a working phase. Imagine that Alice has an interest in whistleblower websites

such as WikiLeaks. During the learning phase she will be grouped with users with similar fingerprints which have either accessed Wiki-Leaks or exhibited a related traffic pattern. During the working phase the subset of online users $\mathbb{O}$ will provide cover traffic routed to an Anonymizer that will send Alice's posts to the WikiLeaks web servers. In contrast to Buddies, TASP has no threshold for user churn effectively allowing for the deanonymization of clients thus dropping its strong anonymity requirement. Whilst TASP's authors argue that the performance of their technique can outperform Buddies, incurring into a reduced latency and achieving higher throughput for online posts, the reality is that it does not offer long term protection against a state-level adversary nor it offers an elegant solution for the usage of user pseudonyms.

In order to mitigate statistical disclosure attacks, it is necessary to preserve the following invariant: Let Alice be one of the users in a fixed anonymity set $\mathbb{K}$ and $\#\mathbb{K} = k$. The probability of Alice accessing WikiLeaks at any given round r_i must equal $1/k$. In Buddies, it is assumed that if the filtered anonymity set P_i does not change over all rounds i when accessing a specific Nym N, statistical disclosure is mitigated. Picture Alice and Bob having been included in every round's P_i when Alice's Nym N_1 is to be accessed. Under statistical disclosure of N_1 the adversary must assign identical probabilities to Alice and Bob owning the Nym. That is, if for every round i, it holds that $(A \in \mathbb{P}_i) \Leftrightarrow (B \in \mathbb{P}_i)$, then Alice and Bob are probabilistically indistinguishable from each other, hence equally likely to own N_1. TASP also ensures that the same properties are achieved. Assuming the anonymity set in TASP does not change during the working phase each user has the same probability of accessing any destination. Both these systems only falter when the churn in the membership set $\mathbb{K}$ changes over time. In other words, with user churn and mutable anonymity sets under long term passive analysis we can observe that Alice accesses a specific website a substantial amount of times when compared to her peers.

3.6 Discussion

As presented above, Tor is vulnerable to three main forms of traffic analysis attacks: (i) traffic flow correlation, (ii) intersection attacks and (iii) statistical disclosure. Although the work of Nunes solves the first [19] it falters against the last two. Buddies and TASP present solutions for intersection attacks and statistical disclosure, but so far no onion routing network has implemented the algorithms featured in these systems. Moreover, the enhancement of real-world anonymity systems to integrate defensive strategies such as Buddies' and TASP's, gives rise to several interesting challenges such as the creation of churn handling algorithms or balancing strong anonymity, high throughout, or lower latency [72].

To the best of our knowledge, we are the first to tackle the threats of intersection attacks and statistical disclosure attacks for Tor considering a global passive adversary. For securing against traffic correlation attacks, we borrow the ideas from TorK, so that when a client accesses a certain web domain any of the k users in the anonymity set might have been equally responsible for initiating a circuit. Further, to resist against intersection and statistical disclosure attacks, we will require the usage of fixed k users' anonymity sets accessing, with equal probability, a set of m predefined websites, whilst preventing any form of churn. We plan to achieve this by developing an add-on system to Tor that does not require the

modification of existing Tor protocols or software components.

Summary

This chapter addresses the proposed solutions to mitigate traffic correlation attacks, intersection attacks and statistical disclosure. For the first the main approaches rely on preventing users from selecting unsafe relays or autonomous systems entirely, for the latter two on the utilization of a persistent anonymity set of users and a threshold for the minimal amount of users tolerated in a group. Furthermore, we detail the most important works providing efficient resistance mechanisms against these attacks. Finally, we discuss the drawbacks of each solution and how to augment TorK using the techniques described for strengthening anonymity sets. In the following chapters we are going to present the design of Moby, our proposed system to enhance TorK against intersection attacks and statistical disclosure.

Chapter 4

Design

In this section, we present the design of Moby, our proposal for enhancing the security of Tor against a global passive adversary. We start by providing an overview of our system, and then present how we propose to overcome several technical challenges of materializing it on top of the existing TorK network.

4.1 System Overview

To offer Tor users stronger anonymity guarantees, we introduce the concept of *k-anonymous flashmobs* (henceforth simply referred as *flashmobs*). Drawing this analogy from real world flashmobs, the idea is to allow a Tor user to schedule an event where a community of other $k-1$ users (designated *mobbers*) is summoned to connect to the Internet, around the same time, and generate covert Tor traffic that will allow the flashmob organizer to "blend in with the crowd" and access an intended website through Tor with k anonymous protection. Similar to TorK, flashmobs will also ensure traffic pattern indistinguishable, thus protecting against traffic correlation attacks. However, differently from TorK, flashmobs will also prevent intersection and statistical disclosure attacks by forcing the set $\mathbb{K}$ of k participants and the set $\mathbb{M}$ of m accessible websites to be locked, preventing these sets from changing for each specific flashmob. If a flashmob is instantiated multiple times in the future, this invariant ensures that the anonymity sets will always remain constant. As a result, a global passive adversary will not be able to erode k-anonymity by taking advantage of anonymity set fluctuations.

To materialize this idea into a concrete flashmob service for Tor users, we present a system named Moby. Represented in Figure 4.1, Moby itself consists of two main components: *bridge* and *client*. Taken together, these components implement a new pluggable transport for Tor. Bridges are responsible for managing flashmobs and relaying the Tor traffic generated by the client software running on users' devices. Bridges are interconnected to enable the enrolment in the same flashmob of a large number of clients, possibly connecting from various regions of the world. Clients run a software bundle comprised of two subcomponents: *gateway* and *agent*. The gateway regulates the transmission of Tor traffic between the local client and the bridge. The local source of this traffic can be either a standard Tor browser or an agent: the former is used by the flashmob organizer to access the intended website, and the latter is

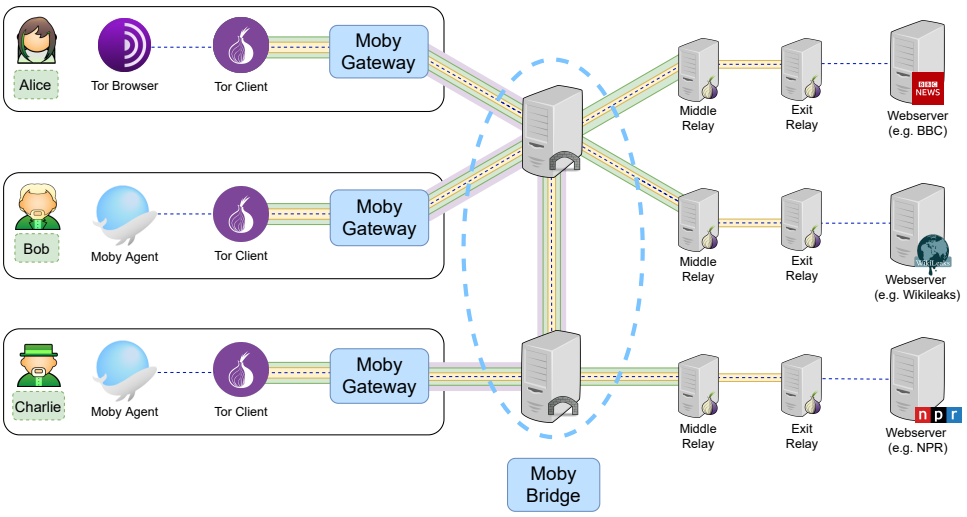


Figure 4.1: Moby architecture. Alice, Bob, and Charlie form a flashmob that can access BBC, NPR, or WikiLeaks websites. A global adversary can observe the network traffic exchanged by every node. Each gateway is manages traffic generated by several Moby users.

used by a mobber for generating covert traffic in the background every time the flashmob takes place.

To illustrate these concepts, imagine Alice wants to access WikiLeaks and post incriminating information about her national security agency employer – a notorious global passive adversary – whilst covering her tracks. She will only connect to Moby when wanting to leak information. In other words, Alice will not use Moby for regular everyday website accesses but only sporadic whistleblowing actions. As such, in our design, Alice is able to connect and engage with trustworthy freedom-loving participants to help her process of blending in with a crowd. Figure 4.1 illustrates how Alice can publish this content while covering her tracks with the help of two mobbers: Bob and Charlie. At a predefined time scheduled by Alice, these three participants must connect to Moby's bridges. When the flashmob starting time is reached (all users are online), Alice, Bob, and Charlie will access the web servers of BBC, NPR and WikiLeaks. Whilst Bob and Charlie are merely aiding Alice by providing cover traffic (generated by the local agents), Alice will post relevant whistleblowing information instead, using her Tor browser instance.

By requiring our fixed membership set $\mathbb{K}$ of k mobbers, to access the same fixed set $\mathbb{M}$ of m websites we are allowing for two specific defenses. (i) First, by requiring every user to remain online over multiple flashmob rounds r_i we will essentially resist every intersection attack. Let $\mathbb{O}_i$ be the set of online users for some round i, if this set remains immutable over all rounds r_i we resist any intersection attack. In other words if $\mathbb{O}_i = \mathbb{K}$ no global passive adversary can infer a whistleblower's identity because the intersection of all online user sets $\mathbb{I}$ equals $\mathbb{K}$. (ii) Second, by requiring every user to access every website in $\mathbb{M}$ we make it impossible to statistically disclose Alice's identity. Let k_i be a randomly picked user from $\mathbb{K}$ and m_i a randomly chosen site from $\mathbb{M}$. The probability p_{k_i,m_i} of k_i accessing m_i essentially equals $\frac{1}{k}$. In other words, the probability of any of k accessing a random website of m is the same. Next, we discuss some important design challenges we need to overcome when building Moby.

4.2 Managing Flashmobs

To provide a k-anonymous flashmob service for Tor, we need to characterize the stages that constitute the life cycle of a particular flashmob. We also need to specify how various actors will be engaged at each stage and what their expected behavior will be. In addition, we need to determine the meta-data required to coordinate the flashmob throughout its life cycle. From this analysis, we will draw the necessary information to design the security protocols responsible for managing flashmobs in our system. Based on our preliminary study, we present our first insights on how flashmobs will be managed in Moby.

4.2.1 Flashmob life cycle

Taking Alice's scenario as example, our preliminary protocol for bringing a k-anonymous flashmob into existence consists in four stages:

1. *User registration:* Initially, Alice will sign up and log into the system. Signing up is an anonymous action and only Moby will know her IP address. She is attributed a secret and transient token that serves as a user identifier. Alice logs in by establishing a connection with a bridge. Alice can switch the bridge she connects to at any time during her session's lifetime.

2. *Flyer generation:* Next, Alice announces her intention to organize a flashmob and creates a *flyer*. A flyer is a manifest that establishes the size of the flashmob k Alice wants to blend into and the list of websites necessary to provide cover traffic m. During this generation phase, Alice is able to hand pick which users she wants to include in her flashmob. Several recruitment policies are possible, e.g., Alice can invite mobbers from a pool of "freedom fighters" willing to help cover her tracks for free or in exchange for money.

3. *Event scheduling:* During this phase, Alice will schedule the performance of a flashmob for a specific period of time. Let t_i and t_f be the initial and final timestamps of a performance, it is expected for every mobber to be present throughout the performance until its completion. As such, Alice will also establish a waiting room where the system will wait for every mobber to be present before starting the flashmob. Let t_{wr} be the starting time instant for the waiting room, it is expected for $t_{wr} \leq t_i$.

4. *Flashmob gathering:* Lastly, the flashmob must be put in motion. Every mobber is expected to connect to the waiting room between t_{wr} and t_i. If a single one of the k users is not online by t_i, the flashmob will be canceled. At moment t_i every mobber k starts the cover traffic generation. Reciprocally, at t_f every one of the k mobbers will cease traffic generation. During t_i and t_f, Alice will get to access to a sensitive-content website of her choosing without a global passive adversary being able to pinpoint which of the k users is accessing any of the m websites. Assuming there is no churn on the anonymity set over every flashmob round r_i, no global passive adversary can deanonymize Alice. Moreover, if k users keep on accessing m with equal probability, we nullify any statistical disclosure attack to our flashmob.

4.2.2 Flashmob flyer

The flyer for a flashmob is constructed as the four phase flashmob creation protocol is run. Figure 4.2 depicts the flyer advertising Alice's flashmob. Having broadcast her intention to create a flashmob in phase 2, Moby will define a unique identifier as the flashmob's name. Furthermore, this is the phase where Alice hand picked her colluding mobbers generating her anonymity set. The system registers the mobbers in a set consisting of their user identifiers. Having chosen BBC, NPR and the WikiLeaks websites for set $\mathbb{M}$, these are also represented in the flyer in the next field. During phase 3, Alice scheduled the duration of every flashmob and the dates for their performances. Alice chose three different UNIX time dates and defined the performance and waiting room periods, for every performance, to be 600 and 1200 seconds respectively. Every user in the system possesses an updated version of this data structure. Every update to this data structure is broadcast to users pertaining to flashmob 1024.

4.3 Managing Users

We need to overcome several challenges involving the management of users, especially dealing with churn and mobber recruitment. We discuss them briefly.

4.3.1 Dealing with churn

One of the main concerns with our system, naturally, is mobber churn. In Moby, differently from other systems [22, 23], churn is by no means tolerated. If a user does not show up for a flashmob gathering at time instant t_i, the flashmob is canceled. However, the incentive and usage model Moby users fall into, differs from those of other systems, such as TorK[76]. Mobbers, for the most part, are passive users that are idle in the background generating traffic and willing to help a whistleblower blend in with a crowd. For example, news agencies, or freedom activists could have a network server or device running Moby on background. As such, the risk of users connecting and disconnecting as they perform their everyday online routines, enabling an intersection attack, is largely reduced. However, there is still the risk that users do not connect to the waiting room during t_{wr} and t_i or disconnecting between t_i and t_f. Next, we also discuss how we propose to tackle this issue.

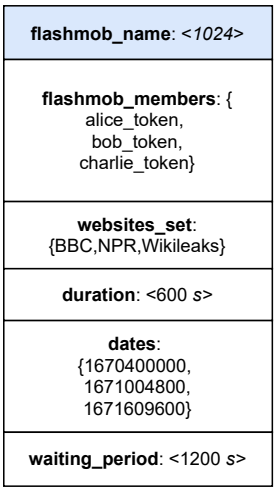


Figure 4.2: Flyer advertising flashmob #1024 proposed by Alice.

4.3.2 Mobber recruitment

Moby allows for the hiring of individuals to increase the size of anonymity sets. Fundamentally, we propose two main methods for hiring additional mobbers: (i) the usage of monetary compensation where each participating user would be rewarded with some form of stake, such as cryptocurrency [77, 78]. As such, every mobber would essentially be mining when participating in a flashmob. (ii) The usage of gamification where users would grind reputation for different factions that simulate distinct system-set traffic patterns. As such, a user would have the desire to grind rating through flashmob participation as a way to display trustworthiness to potentially mobbers looking for a group [79, 80]. It is worth mentioning that both approaches are not mutually exclusive. Furthermore, in order to prevent mobbers with malicious intent to join the waiting room and leave in the midst of a flashmob gathering, we devised a proof-of-stake solution to incentivize users from abruptly leaving. Since every user is compensated with stake in the system, every time a mobber joins a flashmob they essentially put their stake into that flashmob gathering. If they disconnect during the period of the flashmob event or do not show up either stake is taken from them and redistributed to mobbers who have shown up, or they lose flashmob participation rating revealing them to be less trustworthy to peers.

4.3.3 Resisting Sybil attacks

An adversary with significant computation power might own several devices acting like malicious mobbers, a realistic scenario in modern times with surveillance agencies creating bridges in anonymity networks [81]. This would decrease the anonymity of a set or even allow to infer the identity of a user (if all the mobbers in a flashmob are sybil actors). To address this problem, Moby currently allows for “closed” groups where every mobber is invited by the flashmob organizer therefore establishing a trusted group. Each user would be defined by a static-roster of identifier tokens listing all members. A downside of this approach is that it reduces the chances of finding mobbers to participate in flashmobs. To support “open” groups we could build upon Sybil-resistant schemes such as those based on social networks [82, 83] – or limit the rate of users joining the system via some “barrier to entry” e.g., requiring users to solve a CAPTCHA or requiring a phone number to “sign up” into the system.

4.4 Managing Flashmob Traffic

To ensure that our system is robust against both traffic analysis attacks and intersection/statistical disclosure attacks, we need to take particular caution at how the participants of a flashmob generate traffic. We discuss two main aspects involving traffic shaping and generation of mobbers’ dummy requests:

4.4.1 Traffic shaping

One of the main features we retain from TorK is the ability to shape traffic and resist correlation attacks. In TorK, clients and bridges implement a traffic shaping protocol that obfuscates the real patterns of the Tor payload traffic in such a way that the k participants of a k-circuit are indistinguishable from each other in the eyes of a global network eavesdropper. This is an essential characteristic to preserve k anonymity that we also borrow into our system. In Moby, the participants of a flashmob connect to a bridge and all the traffic that they generate will be modeled according to the same traffic shaping function.

Another key feature inspired by TorK is throughput leveling. This feature is used to prevent the correlation of TorK traffic with the Tor circuit by observing varying bursts in the traffic of clients. The main idea is the usage of a contiguous traffic flow by each client when accessing destination websites. For instance, two Tor cells received by different clients within the same k-circuit having different uplink capacity (e.g. 200Kbit/s and 1Mbit/s) are drained to the Tor network at the speed of the slowest client. In this way, an attacker cannot correlate a client to its Tor circuit based on timing properties even when delaying clients connections deliberately. Likewise, in Moby we will need to implement a similar technique to ensure that the traffic exiting the bridges into the Tor network is homogeneous across all the participants in a flashmob.

4.4.2 Mimicking realistic browsing

Every time that a flashmob takes place, the $k-1$ participating mobbers helping the flyer creator to attain k-anonymity, need to generate HTTP requests to the web sites indicated in flashmob's flyer. To preserve robustness against traffic analysis attacks, it is necessary to generate realistic website accesses, otherwise a global passive adversary might notice the automatic generation of cover traffic by pattern inference. Lorimer *et. al.* have implemented a web-based traffic simulator that browses the web as a human would [84]. We propose to leverage this work to help us emulate realistic cover traffic generation. Furthermore, our system also allows for mobbers to manually access a website of their choosing, provided it is part of the $\mathbb{M}$ websites list advertised by the flashmob creator on the flyer, and the access patterns do not leak information that may be leveraged to launch statistical disclosure attacks. This would give mobbers extra incentives to engage in flashmob events.

4.5 Managing Bridges

Lastly, we discuss design challenges involving bridges, namely how to guarantee their proper coordination and detect the deployment of malicious bridges.

4.5.1 Bridge coordination and load distribution

In Moby, we build our bridge infrastructure by extending the original bridge architecture proposed in TorK by interconnecting bridges with each other forming a global mesh architecture. Each of these bridges will distribute traffic across different middle relays of Tor. In turn, these middle relays will route traffic to exit relays and their destinations. Differently from TorK, we assume a global network of users, as such mobbers can be connected to different bridges across the globe and still pertain to the same flashmob. As a result, bridges will also include mechanisms to securely share potentially sensitive metadata about users or flashmobs.

Users can choose which bridge they want to connect to and authenticate with the unique and transient identifier attributed to them upon registration. For load balance purposes, when users connect to a bridge the traffic may be routed to several middle relays. Differently from TorK, Moby is designed under the assumption that, once a flashmob is in its course – i.e., all the participants are currently online and generating traffic as specified in the flashmob's flyer – the users currently joining the flashmob are stable, i.e., no new users are allowed to join. As long as there are k connections to a bridge these will persist until the end of the flashmob event. This way we keep on ensuring k-anonymity whilst providing resistance against intersection and statistical disclosure attacks.

4.5.2 Ensuring the correct execution of Moby bridges

Similar to TorK, we will also need to deal with the possibility of malicious bridges trying to infiltrate into Moby's bridge infrastructure and compromise the security of the system, e.g., leaking information about the actual source and destination of the Tor circuits going past the bridge. To protect against these attacks, we also inherit TorK's ideas of securing bridge state using Intel SGX enclaves [85], and verifying the software integrity of ingressing bridges through hardware-based remote attestation.

Summary

This chapter detailed Moby overall architecture, main entities and threat model. Next we described the flashmob concept, their proccess of coordinating k-users and ensure resistance against intersection attacks and statistical disclosure. Next, it is discussed measures to deal with user churn, the recruitment of users to a flashmob performance and resistance against Sybil attacks. Following, we the thwarting of correlation attacks and possible strategies of emulating realistic participant actions when accessing websites. Lastly, it was discussed measures against malicious bridges and clients which can be deployed and launched by an attacker.

Chapter 5

Implementation

This chapter addresses the implementation details of our Moby prototype. Section 5.1 describes an overview of the implementation. Then, we provide several details about the implementation of flashmobs in Section 5.2. We address the Moby CLI used to schedule flashmobs in Section 5.3. For the real-world deployment of Moby we elaborate on the hardware that was used on Section 5.4. In Section 5.5, we address how the devices were monitored. Section 5.6 refers to the plug-and-play implementation of Moby. Lastly, Section 5.7 demonstrates how we got Moby to properly work behind residential networks.

5.1 Implementation Overview

In this section, we brief the reader on the main components of our work. We will mostly focus on how the Moby add-on interacts with the components of TorK, how does TorK leverage Tor, and how a client can use Moby.

Figure 5.1 depicts the major internal subcomponents of the Moby add-on and the TorK implementation, which comprises hub and bridges. To route Tor traffic through TorK, the hub must be set up to use the TorK pluggable protocol and run Tor. Upon starting, Tor spawns the TorK hub service, the Moby add-on and a SOCKS [86] proxy to receive Tor traffic. Henceforth, Tor cells generated by the client can then be received by the hub service, packed into TorK frames and sent to the bridge according to a specific traffic shaping function. Conversely, TorK bridges extract the Tor cells embedded into TorK frames and deliver these cells to the Tor network.

Since TorK is a Tor pluggable transport, any TCP/IP application that places requests through Tor can reap the benefits of our system. To entirely control a Tor circuit, TorK instrumentally reroutes streams to a given circuit only when authorized by bridges. To perform this rerouting operation, we leverage the Tor Controller specification [87] to build a Tor controller that exposes functions capable of creating, closing, and attaching streams to circuits.

Once initialized Moby, will leverage a CLI interface allowing the user to send commands to the bridge the client is currently connected to, and create a flashmob that future users, once connected to this same bridge might be interested in joining. The Moby component for the bridge not only receives commands

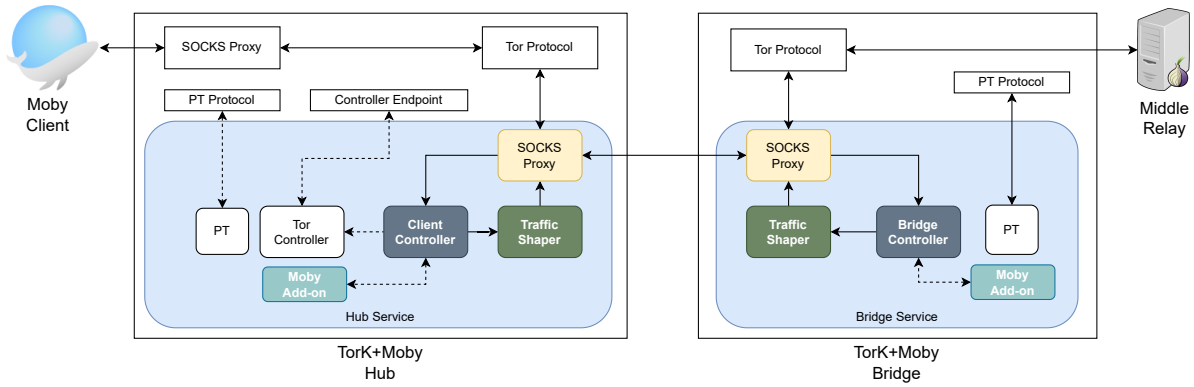


Figure 5.1: Overview of Moby's components with TorK

to instantiate a flashmob, but is the responsible process to coordinate mobbers to start traffic generation and block traffic if the number of messages does not match the total number of users for the instanced flashmob.

We implemented our Moby prototype on top of TorK for GNU/Linux. Moby includes both a Client and Bridge (Server) component. The prototype was written in C++ using OpenSSL[1] and Boost[2] libraries.

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

5.2 The Moby Add-On

As an Add-on, Moby extends the functionalities offered by TorK, either by including extra-commands that allow for the coordination of clients or by implementing security measures, such as minimum thresholds for the number of connected users, intended on mitigating intersection and statistical disclosure attacks. In other words, Moby leverages existent functionalities implemented by TorK such as setting the minimum number of users required to instance a k-circuit – blocking traffic routing if the current number of connected users is less than k – and adds coordination features to deploy mobbers when a flashmob is happening.

Moby's protocol is based on text streams which can be implemented by TCP sockets. Usually, all messages exchanged from the Moby client and bridge modules are replies to mobber commands previously sent. Moby also includes assynchronous commands which are sent from the bridge to clients when a flashmob is scheduled to start or terminate.

Moby is implemented on the bridge by two threads. The main thread, is responsible from receiving Moby commands, such as the creation of a flashmob and storing in a hash table for the purpose of retrieving the flashmob data structure by identifier once a flashmob is scheduled to begin. The flashmob

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

name, introduced in Section 4.3 is used as the hash table key, and the hash table value is a data structure implementing the “flyer” fields. The other thread, is the “Scheduler” thread, being used solely to check on the dates of the flashmob flyers stored in the hashtable. Once the current clock matches the dates of flashmobs stored in the hash table, the “Scheduler” thread will instantiate a temporary sub thread to monitor the current flashmob. These “Scheduler” sub threads will constantly uphold more specific fields such as checking the number of users connected during the “waiting period” and “flashmob duration” – picking the action of starting and ending flashmobs based on these connections. Once a flashmob concludes, this thread is responsible for cleaning the current date from the flyer.

On the other hand, the client sends commands to the bridge through the “Client Controller”. Moby’s Add-On extends the possible commands that can be used by a mobber. Once Tor starts running, it will instantiate as a sub process the main Moby thread. This thread is responsible for receiving commands from the Moby bridge “Scheduler” sub thread. Specifically, it will receive the asynchronous commands to be run on the “Client Controller” leveraging TorK to configure the current Tor streams. It is important to mention that the size of the *k*-circuit is still configured by the client, similarly to TorK. This happens because it allows the Moby bridge to count the number of connections during the waiting room and order the correct size of *k_min* to be selected by the mobbers.

5.3 Moby Controller Interface (CLI)

Moby exposes a controller to configure the flashmob “Flyer” and tune TorK settings such as the size of the *k*-circuit being used by a flashmob. Moby essentially extends the functionalities of the TorK text based stream protocol (CLI). Moby allows users to select the flashmob participants, the websites being accessed, the duration of each round, the dates for future rounds, and the waiting room period as shown in the “Flyer” in Section 4.3.

CLI is implemented as a simple socket server, receiving commands, passing them to the Bridge/-Client Controller handler which take a certain action depending whether we have a Moby bridge or Moby client gateway, and processes a reply back to the user. CLI is used primarily by clients. We also developed a Bridge CLI which was used for flashmob testing functions (such as manually start or end a flashmob). For instance, the main Moby CLI commands available are:

- `flashmob_name`: defining the identifier of a flashmob
- `flashmob_users`: gives a list of the IP addresses of the possible connected mobbers
- `flashmob_websites`: defines a list of websites to be randomly accessed
- `flashmob_duration`: sets the duration of the scheduled flashmob
- `flashmob_dates`: schedules the list of dates for future performances
- `flashmob_waiting period`: sets the duration for the waiting room period

(a) Top view of the RPi4 deployment

(b) Side view of the RPi4, notice the numbering labels on the case

Figure 5.2: Pictures of the actual deployment of RPi4 running Moby flashmobs

5.4 Venturing into the Internet of Things (IoT)

Since the main cause of deanonymization for Moby is the churn of flashmob participants, to protect Alice in the advent of sensitive website accesses, we have performed a practical deployment of Moby using Raspberry Pi (RPis) microcomputers acting as Moby hubs. These devices are intended to function as mobbers being deployed behind the residential network of Moby users and generating traffic once a flashmob begins.

The reason behind this deployment was that, differently from desktop, laptop computers and mobile phones, Internet of Things (IoT) devices such as RPis are meant to be online during long periods running relatively simple, non-intensive computational tasks whilst being available almost 24/7. Differently from computation servers, RPis offer low energy consumption and are practical to deploy, in an almost plug-and-play fashion.

Through the usage of RPis, deployed in residential homes, intended to be online and running Moby client 24/7, the availability of mobbers is highly increased. In comparison, assuming we would have run Moby on a personal desktop, laptop, or mobile phone the availability might not be the same. This is due to the fact that clients often do not leave their devices up and running and, as such, even if having signed for a future flashmob, a client might be offline during the time period of the flashmob performance. Lest the occasion of power outtages and network failures the probability of churn resides on the client willingly quitting the flashmob the node was signed for.

Although offering many advantages and being quite simple to tinker with, we went through several phases of testing. Initially we experimented with Moby on a RPi3 Model B, which posed several issues. The main issues of this implementation were the hardware specifications of the third version of Raspberry Pis. Not only was the clock frequency quite low at 1.2 GHz, but the RAM offered was 1 GB. After conducting some experimentalMoby runs we concluded that the performance of the RPi was quite poor. Not only it had trouble instantiating Docker machines with only 1 GB of RAM, but when using Moby

Figure 5.3: Statistics for one of the deployed RPi devices for the entire duration of August, 2022

through the command line, accessing a website with curl[3] was quite slow from an usability standpoint.

As such, we redeployed Moby for the RPi4 Model B version of these micromputers. Not only did this version offer a higher clock rate at 1.5 GHz, but also higher RAM possibilities – 1 GB, 2 GB, 4 GB, and 8 GB. Although testing Moby with a RPi4 with 4 GB of RAM, either with Docker or native through the command line, we ended up choosing the RPi4 Model B with 2 GB of RAM. The reason for this, was that the performance in both applications was quite close from an usability stand point and the 2 GB version was widely accessible on consumer electronic stores.

As such, after having picked the RPi4 (2 GB) and comparing the performance to the RPi3 (1 GB), the Moby experience was enhanced because, having more RAM we could easily run several Moby clients on docker containers within the RPi4[4]. In the end, we ended up running Moby without the usage of virtualization or deployment software since these features would penalize the performance of Moby.

Once the RPi4 devices were deemed sufficient and able to run Moby flashmobs, a batch of these micromputers was ordered from electronic retailers. It did not help that, as of the year of writing, we are in the middle of a semiconductor shortage. Not only did this shortage affect the Raspberry Pi production [88], but finding a store capable of delivering a batch of these devices within a reasonable deadline was a feat in itself. Even though the experimental setup was delayed, at the end with ended up getting our RPis from a major retailer which received a new batch in August.

Figure 5.2 shows the deployment of RPi devices ready to be distributed through the members of our research group. These devices would later be used for throughput testing of Moby flashmobs in Section 6.3.

5.5 Monitoring the Pis

As the RPis devices were to be deployed across different locations, we would require a way to monitor these microcomputers remotely. As such, we leveraged technologies such as Grafana[5], InfluxDB[6], and

[3]https://man7.org/linux/man-pages/man1/curl.1.html
[4]https://www.docker.com/blog/happy-pi-day-docker-raspberry-pi/
[5]https://grafana.com/
[6]https://www.influxdata.com/

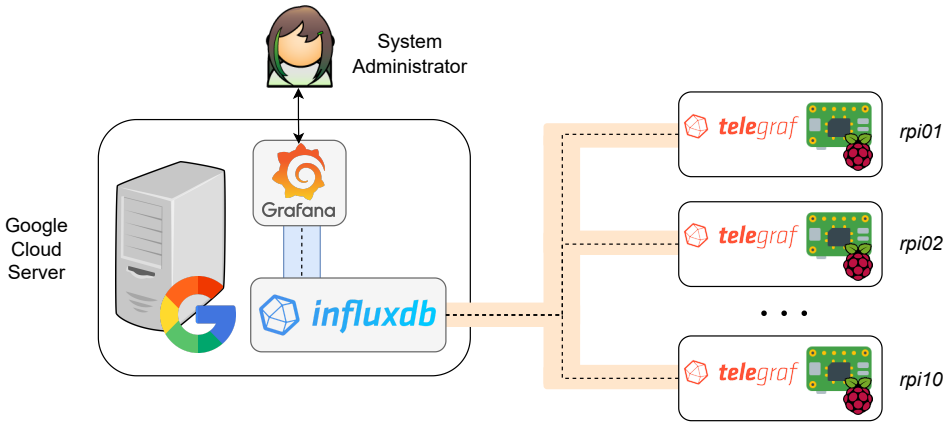


Figure 5.4: Monitoring of the RPi devices through InfluxDB and Grafana

Telegraf[7] for this purpose.

Grafana was deployed on a Google Cloud [89] machine, which would essentially act as the main hub for gathering statistics from every RPi device. Since Grafana works as a interactive visualization interface for a multitude of systems and already has dashboard profiles meant to be used for monitoring of IoT devices, it served as the perfect fit for our deployment. With the Grafana application and web client installed on our machine, the stage was set for having the RPis send their data to this hub.

InfluxDB was deployed in the same Google Cloud machine with the intent of being the main database responsible for gathering the statistics to be displayed in Grafana. As such, every RPi device was configured to run a Telegraf script meant to, every 10 seconds, send updates to the Google Cloud InfluxDB database of each microcomputer CPU temperature, memory and network usage, swap space, number of active processes and threads.

Figure 5.3 displays the graphical user interface used to remotely monitor the microcomputers deployed behind the residential network of Moby users. Figure 5.4 displays the network architecture of the monitoring web interface. The Grafana and InfluxDB applications are instanced on a Google Cloud server located in Frankfurt, Germany (europe-west3). Each RPi from *rpi01* to *rpi10* will send updates of resource utilization, recurring to Telegraf, to the InfluxDB server. The Grafana IoT Dashboard will connect to the InfluxDB database, holding an entry for every RPi and instantiate a dashboard to monitor each of the devices.

5.6 Plug-and-play implementation

Considering the RPis devices were to be deployed behind user networks, we had to find a way to reach them, either for remotely operating the devices or for the scheduling of flashmobs. Initially, we started using Reverse SSH Forwarding. That is, each RPi would establish a SSH connection with a Google Cloud server intended to host the reverse SSH connections of every RPi and establish a tunnel on a different port for each device. This approach lead to several practical issues: (i) once the RPi was disconnected, the connection would not be re-established with the server, (ii) only the ports that were

[7] https://www.influxdata.com/time-series-platform/telegraf/

forwarded were able to route traffic.

The first problem mentioned above posed the biggest issue for Moby. We wanted that, once configured, the RPi could be deployed in any client home and establish a SSH tunnel with the Google Cloud machine so that traffic could be routed through the server and behind the NATs of Moby users. To solve this we utilized *autossh*[8] reverse port forwarding. This program is implemented as a *systemctl* service and it allows to, once the *autossh* monitor detects the current SSH connection to be down, to reestablish it with the Google Cloud host.

The second problem mentioned was dealt with the establishment of several connections reverse SSH connections with our Google Cloud server. Each connection would create an SSH Tunnel in a different port. As such, this allowed to reverse SSH into any of the devices that were placed behind user NATs. Although this initially worked to remotely operate each RPi, the biggest problem was that some of the SSH connections were quite unstable, and even when using *autossh* it was common for a RPi to be down. Furthermore, not only we wanted to administer the machines, but we also had to use these forwarded ports for traffic being routed through Moby.

As such, this approach was deemed cumbersome and unpractical. Thus, we performed some adjustments in our approach for making Moby to be functional behind the Network Address Translation (NAT) of residential routers. In order for the Moby bridge to instruct mobbers to start generating cover traffic, these devices have to be reachable from outside the NAT. For enabling this task, we have made use of Tailscale [90] – a plug-and-play secure VPN service – allowing mobbers to expose a public virtual IP address accessible to the Moby bridge.

With Tailscale, every user will now expose an extra virtual network interface to the outside world. These virtual network interfaces will act as a regular interface, thus giving these machines a reachable IP. Not only that, but it facilitates reaching any of these devices, no matter where they are located, let the goal be to SSH into the RPis or use them for a distribution application such as Moby.

Nevertheless, it is important to mention that, although the Moby bridge used the Tailscale IP addresses to contact mobbers, once the flashmob starts, the client nodes will route traffic regularly (through the bridge and relays). As such, Tailscale is used exclusively for the process of broadcasting flashmob management instructions (start and finish) to otherwise unreachable nodes behind a home network.

Not only, did Tailscale provide public addresses for mobbers behind NATs, now reachable from Moby bridges outside of this VPN, but it allowed for every device connected to the same Tailscale network instance to also communicate with one another. As such, the members of our research group could simply power the device and connect it to the Internet and ping (or even SSH) into any member of the flashmob. This implementation helped to shape the concept of plug-and-play Moby-ready device.

Differently from reverse *autossh*, which required configuration of *systemctl* background services and would not work properly when changing IP addresses, Tailscale allowed for a simple installation that could be done in a different IP address than the one the Moby-ready device would be deployed to. As such, when connecting the device to a home network router, each RPi would pop-up in the Tailscale administration console, truly bypassing the NAT and being reachable from the outside Internet, including

[8] https://linux.die.net/man/1/autossh

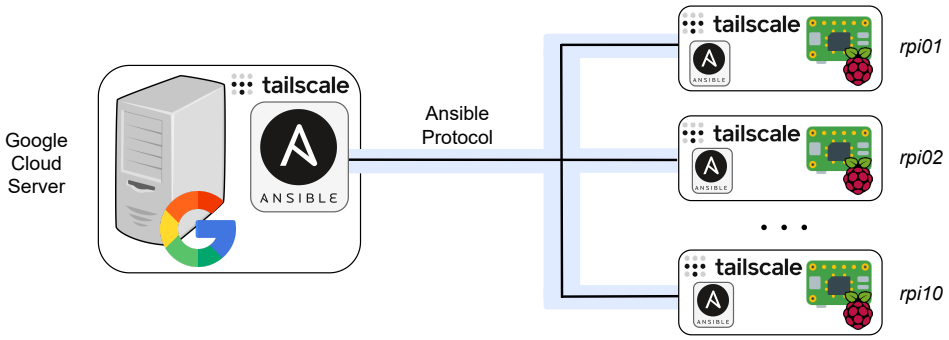


Figure 5.5: Update of RPis through Taiscale using Ansible

members of the flashmob, for users with knowledge of this IP.

5.7 Updating Torklets behind NATs

Having the RPis connected in a virtual network to one another and deployed on residential homes, we had to find a way to remotely operate these devices and provide them with the Moby software stack.

Since adding a new node to the Tailscale network is very easy (one installs Taiscale and opens the resulting *http* link on the browser) it was simple enough to add a Google Cloud machine into the network. This machine would act as an entry gateway into the Tailscale, allowing an user, once connected, to SSH into any of the devices.

The reason we had to add a gateway server to the Tailscale network was twofold: (i) a user can only SSH into another Tailscale node if the current instance is also part of network, and (ii) we needed a publicly accessible server to SSH into. For challenge (ii) the other option would be to have our personal devices also instanced in the Tailscale network, which in our opinion was not a realistic scenario since we wanted to separate the flashmob members from other unrelated devices.

Furthermore, with the need to remotely operate a substantial amount of microcomputers with the Moby software stack and provision other administration needs to the RPis, we had this same gateway node acting as Ansible [91] master node. Since Ansible requires its master node to be able to SSH into any of the Ansible clients, and Tailscale provides a SSH service[9], it was natural to exploit the synergy of these two applications.

Figure 5.5 depicts the components and how they interact in the real-world deployment of Moby. The Ansible master node hosted on Google Cloud provisions the RPis with the software pre-requirements, installation of Tor, TorK, and Moby. Once a flashmob starts the Moby bridge broadcasts the start flashmob command to Tailscale IP of each mobber. The mobbers start the website accesses through the Moby bridge. Every node on the system implements both a Tailscale network interface with SSH enabled, and also acts as an Ansible client. These configurations allow both for instancing flashmobs and providing updates on the Moby system despite any network placement of the mobbers.

Despite updates for Moby nodes being provided with Ansible through a Google Cloud server connected in the same Tailscale network as the RPi devices, we observe these properties as intruding in

[9]https://tailscale.com/tailscale-ssh/

the privacy of users. The usage of Tailscale was needed so that even when a mobber does not have Tor running on its machine, the Moby bridge can reach the client and start an instance of Moby. For future work, Moby should be updated through a packet manager instead of Ansible, and nodes should be responsible in provisioning a IP reachable, even behind NATs.

Summary

This chapter has described the implementation details of Moby prototype. The setup of the Moby Add-on, how it is implemented on top of TorK and the functionalities implemented, mainly scheduling of flashmobs and resistance against statistical disclosure attacks. We also delve into how the real-world deployment of Moby was conceived mainly utilizing the correct hardware devices, the monitoring of their resources, the plug-and-play implementation for Moby devices, and the updating procedure required to deploy the Moby software stack amongst devices behind NATs.

Chapter 6

Evaluation

This chapter describes the evaluation of our system. We start by explaining our methodology (Section 6.1). Then, we study and analyze Moby's performance under different settings (Section 6.2). Next, we demonstrate how a real-world deployment of Moby achieves high-levels of availability for the composition of flashmobs. Lastly, considering instances where our system performs in less reliable environments, we examine Moby's resistance to statistical disclosure attacks under high availability conditions.

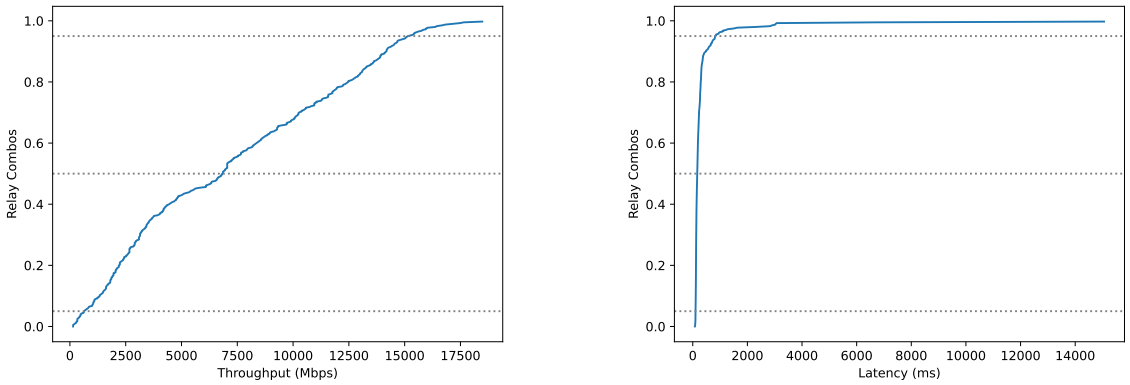
6.1 Methodology

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

6.1.1 Evaluation Goals and Approach

Our main evaluation goals are twofold: (i) assess the performance of our Moby prototype, and (ii) measure its resistance against statistical disclosure. In particular, regarding Moby's resistance against statistical disclosure attacks, we wish to assess how: (i) availability of mobbers protects Alice, (ii) how Alice's absence from flashmobs enables an adversary to profile Alice's probability of accessing sensitive websites, (iii) an oracle implemented on the Moby bridge would affect system performance.

Recent literature aimed at deanonymizing anonymity sets has focused on profiling the websites being accessed by different users within anonymity sets. Typically, existing approaches for statistical disclosure are analytical, such as the original statistical disclosure attack proposed by Danezis [56], the generalized statistical disclosure improvement introduced by Mathewson and Dingledine [57], and the least squares statistical disclosure attack by Pérez-Gonzalez [59]. For our analysis, we leverage the original statistical disclosure attack. We relegate the usage of the generalized and least squares statistical disclosure for future work.



(a) Average Relay Throughput

(b) Average Relay Latency.

Figure 6.1: Performance metrics of the average Tor circuit combinations

6.1.2 Experimental Testbed

Our laboratory testbed is composed of a Tor bridge and 25 clients, where each node runs an instance of Moby (leveraging Tor v0.4.2.8). Our nodes are executed within Docker containers, provisioned either with 2 vCPU and 2GB of RAM in the case of the bridge node, or 1 vCPU and 1GB RAM for client nodes. The bridge and client nodes run in separate physical hosts equipped with 2 Intel Xeon CPU E5506 vCPUs.

Each client was configured to use a specific Tor circuit throughout our experiments. We manually selected all relays comprising each Tor circuit based on Tor's *TopRelays* list [92], and chose different nodes according to these conditions: (i) nodes have to be located in Europe, (ii) have a throughput above 5 Mbps, and (iii) have an uptime above 10 days.

For conducting the above experiment, we build a representative dataset of multiple connections between clients and a bridge, and, respectively, between the bridge and the Tor network. We perform the experiments in two phases: (i) the 25 clients will generate chaff traffic towards the bridge, and (ii) the 25 clients will fetch files with sizes, ranging from 32 MB to 64 MB, hosted in a private server, over Moby.

6.1.3 Metrics

In this section, we define the metrics used to evaluate Moby's performance and its resistance against traffic correlation attacks:

Performance Metrics: To measure performance we studied the throughput and latency for the chunk size of 536 B. The traffic shaper rate function was modified to maintain the same minimum and maximum throughput. Thus, the rate function selected was $Rate(t, n) = 10 \times t \times n$ µs, where t is the number of maximum Tor cells that fit one chunk and n the number of connected clients.

Statistical Disclosure Metrics: For statistical disclosure, the main metrics we have utilized were: (i) the probability of an attacker profiling Alice from her peers, and (ii) the difference between Alice's real website accessing probability and the probability observed by the attacker. Furthermore, we apply these metrics against a generalized availability scenario (i.e. a scenario where failure of nodes ranges from

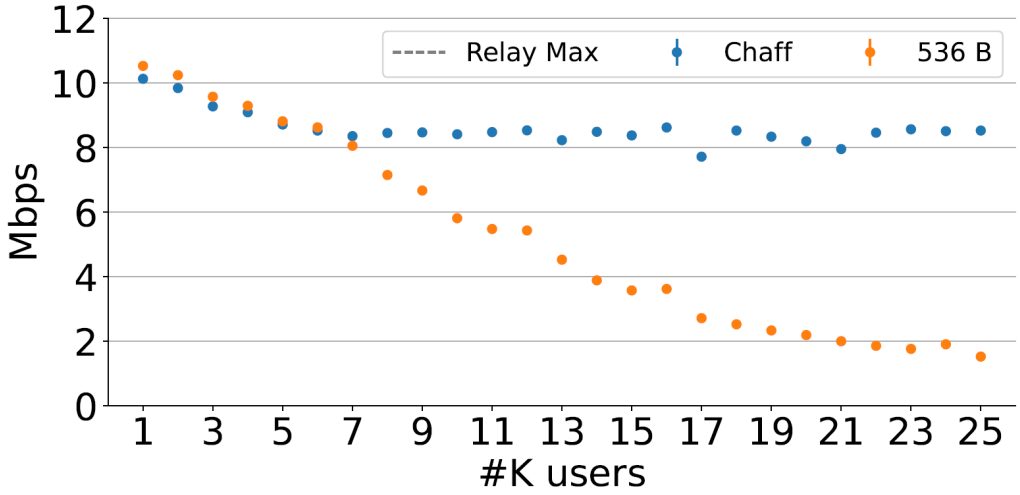


Figure 6.2: Throughput of a Moby flashmob using an average circuit

0% to 100%, in 5% increments), a high availability scenario, an absence-tolerant implementation, and an approach where an oracle featured on Moby bridges has a minimum threshold for the users in an anonymity set, i.e., an implementation similar to Buddies [22] where no traffic is routed if an anonymity set has less than a predefined minimum amount of users.

6.2 Performance

In this section, we evaluate the performance of using different flashmob sizes for Moby. Before performing this evaluation, we sought to assess what the performance for an average circuit is, i.e., by gathering data of multiple possible circuits around Europe, we determined which circuits would hold the most realistic results.

6.2.1 Determining the Circuit

To determine which would be the ideal circuit to run our experiments we gathered information regarding a total of 32 middle relays and 32 exit relays. The conditions for picking a relay were stated in Section 6.1.2. Afterwards, we analysed every possible combination between our chosen middle and exit relays. It is important mentioning that the Moby bridge acts as an entry relay, and as such, it is the only Tor relay that stays the same.

Figure 6.1 displays our evaluation for the latency and throughput for a total 1024 circuits (32 middle and exit relay combinations) using a cumulative distribution function. From this study, we could gather the following conclusions, starting with the throughput: (i) the median is 7.32 Mbps, (ii) the 5% percentile corresponds to 532.43 Kbps, (iii) the 95% percentile corresponds to 14.64 Mbps, (iv) the minimum throughput is 0 (corresponding to circuits that could not route traffic), and (v) the maximum throughput is 19.37 Mbps.

For the latency we gathered: (i) the median is 102.89 ms, (ii) the 5% percentile corresponds to 64.01 ms, (iii) the 95% percentile corresponds to 370.15 ms, (iv) the minimum latency is 52.45 ms, and (v) the

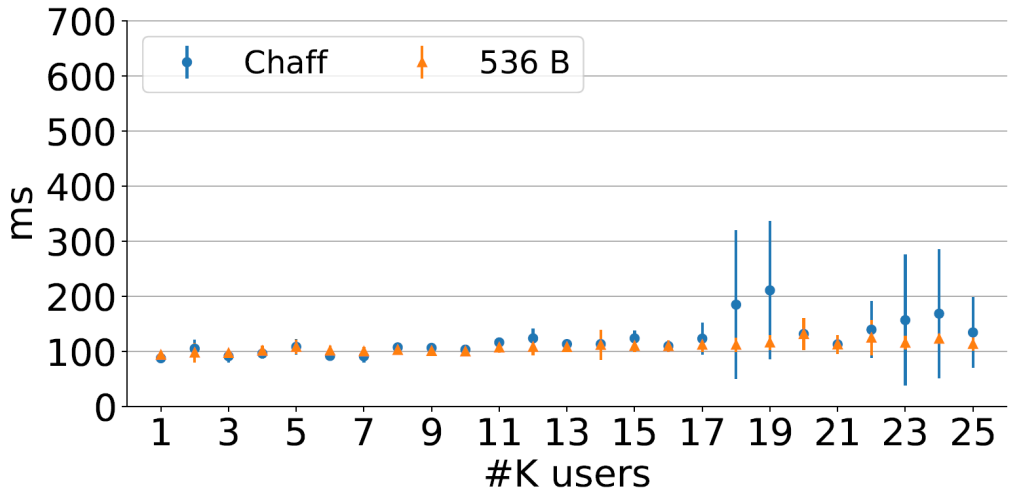


Figure 6.3: Latency of a Moby flashmob using an average circuit

maximum latency is 3203.67 ms.

Having studied the average throughput and latency of 1024 circuit combinations, in the next sections we will evaluate how different flashmob sizes in the Moby perform under a median performing circuit. The reasons for this being: (i) by using a random circuit we will not have a realistic reading of the possible latency and throughput, i.e., we might have very pessimistic or optimistic experiment results that do not reflect reality, and (ii) we need to properly assess which combinations of circuit possess these median characteristics in order to reuse them in the future for several experiments allowing us to have the most neutral test bench to compare different results.

6.2.2 Throughput

To quantify the performance of the system an experiment was conducted to measure the covert channel throughput using a fixed size frame allow us to provide a indistinguishable channel but also will incur in an overhead. We set Moby to use 536 B frames, each frame containing 1 chunk, over a dynamic rate interval $Rate(n) = n \times 10\,\mu s$, where n is the number of connected clients.

To measure the channel throughput, we used IPERF3[1]. The client runs an instance of the iperf3 client through Moby and consequently through a specific Tor circuit performing measurement towards an iperf3 server.

Figure 6.2 depicts the achieved throughput based on the average bandwidth of 3 runs according to the increasing number of connected users K receiving chaff and data. Our results show a client can achieve up to, approximately, 10 Mbps maximum achieved by these relays when using Moby. With a higher number of connected clients, Moby decreases the rate thus reducing the throughput. This reduction is more critical when most clients are, in fact, receiving useful data instead of chaff. This behavior is inline with Moby's expected behavior. Processing chaff frames is less burdensome than data frames since the former are discarded upon reception and the latter require receiving all chunks and deliver the content to Tor.

[1] https://iperf.fr/

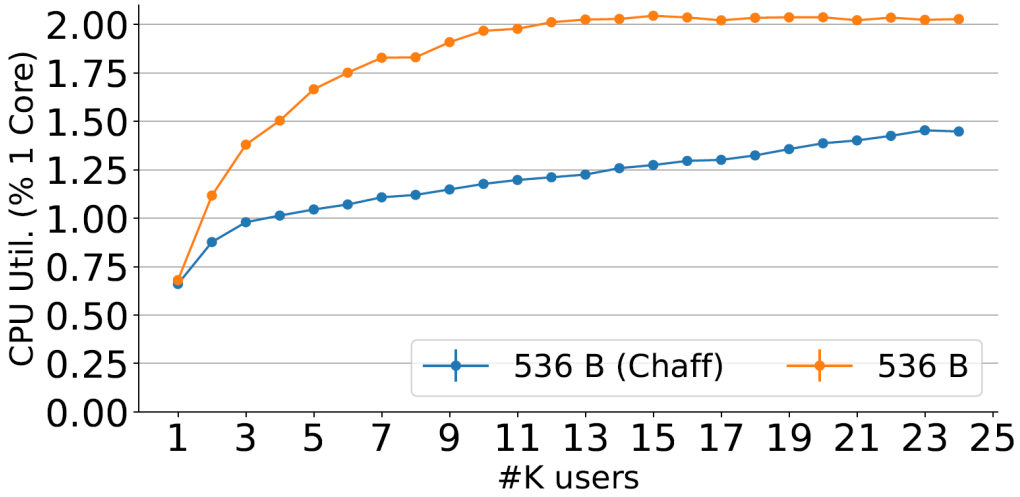


Figure 6.4: CPU usage of a Moby flashmob using an average circuit

6.2.3 Latency

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

Figure 6.3 depicts the average latency of 3 runs according to the increasing number of connected users K receiving chaff and data. An increase in the number of connected users causes the latency to increase. Furthermore, data frames causes slightly higher increases in latency than chaff frames. As one might notice, for a k of 18, 19 and 23 users, standard deviation spikes are greater when compared to other entries, specifically those where k is lesser or equal than 17. It is important to mention that most of these performance measurements were run starting in April 2022 when the Tor network was under constant denial-of-service attacks which resulted in unstable results during our experiments [93, 94].

6.2.4 CPU Usage

Figure 6.4 depicts the bridge CPU utilization in percentage of one core utilization. All experiments were conducted using the hardware limitations of the baseline experiment. Thus, bridges can use up to 2 cores under a 2GB as the chunk size increases CPU utilization decreases due to the traffic shaper rate value. These results show the difference between processing chaff frames and data frames under the 536 B chunk size. Since chaff frames do not need any CPU processing apart from receiving and discarding them, processing chaff frames is a lightweight task in comparison with data frame processing. In comparison, packets carrying actual data require more CPU processing.

Device	ISP	Modem	Wired	Avg. Tput. (Mbps)	Avg. Latency (ms)
rpi01	ISP_1	Fiber	WiFi	**85.51 ± 2.48**	**46.24 ± 1.77**
rpi02	ISP_1	Fiber	Ethernet	**194.96 ± 2.43**	**44.69 ± 0.30**
rpi03	ISP_2	Fiber	Ethernet	**203.43 ± 3.42**	**41.91 ± 1.35**
rpi04	ISP_3	Coaxial	Ethernet	**17.13 ± 2.35**	**49.18 ± 7.26**
rpi05	ISP_3	Coaxial	Ethernet	**21.38 ± 0.82**	**52.07 ± 25.70**
rpi06	ISP_2	Fiber	Ethernet	**117.90 ± 0.84**	**49.01 ± 3.37**
rpi07	ISP_2	Fiber	Ethernet	**92.92 ± 1.94**	**57.58 ± 165.15**
rpi09	ISP_1	Fiber	WiFi	**92.71 ± 0.78**	**118.94 ± 339.71**
rpi10	ISP_2	Fiber	Ethernet	**119.01 ± 0.48**	**42.49 ± 4.91**

Table 6.1: Connectivity characteristics and network performance of the deployed RPi Moby nodes.

6.3 Availability in Real-World Deployment

For a flashmob to be schedule Moby requires every user to be online at the time of the performance. As such, in this section, we present a real-world deployment of Moby using Raspberry Pi4 (RPis) devices. The reason for the usage of these microcomputers is that IoT devices are meant to be running for long periods of time, therefore offering a high availability service, something that we would require in order to mitigate intersection attacks and statistical disclosure.

The microcomputers act as Moby hubs, each provisioned with a 4-core Broadcom BCM2711 CPU and 2GB of RAM. These devices were distributed among different members of our research group and installed in households across the metropolitan area of Lisbon. This distribution enabled us to gather a representative sample of the variety of ISPs and Internet connections' performance expected to be found in the homes of individuals interested in using Moby.

The intended use of these devices would be to mimick flashmob users contributing to the generation of anonymity sets. As explained in Section 5.4 these RPi devices were set up in users residential networks behind the NAT of home modems. With this deployment we intend on measuring the performance of Moby in two dimensions: (i) throughput and latency of each device without usage of Moby, (ii) throughput and latency using Moby flashmobs.

Table 6.1 details the connectivity characteristics of each RPi machine. The table also shows the average throughput and latency obtained by each node over the course of a week-long measurement where throughput and latency statistics were obtained every two hours. These statistics were obtained by reaching out to a public server under our control, hosted within a different European country, using the `httping` and the `iperf3` utilities, respectively. We did not send traffic through Tor or Moby during these measurements, i.e., we intended to measure the raw network performance of these devices. As seen in the table, our measurements reflect an heterogeneous landscape of network performances, where the throughput of RPi nodes ranges from 17 Mbps to 203 Mbps. Interestingly, we see that even the low performance nodes achieve a sufficient throughput to sustain bandwidth-hungry applications, like video streaming.

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

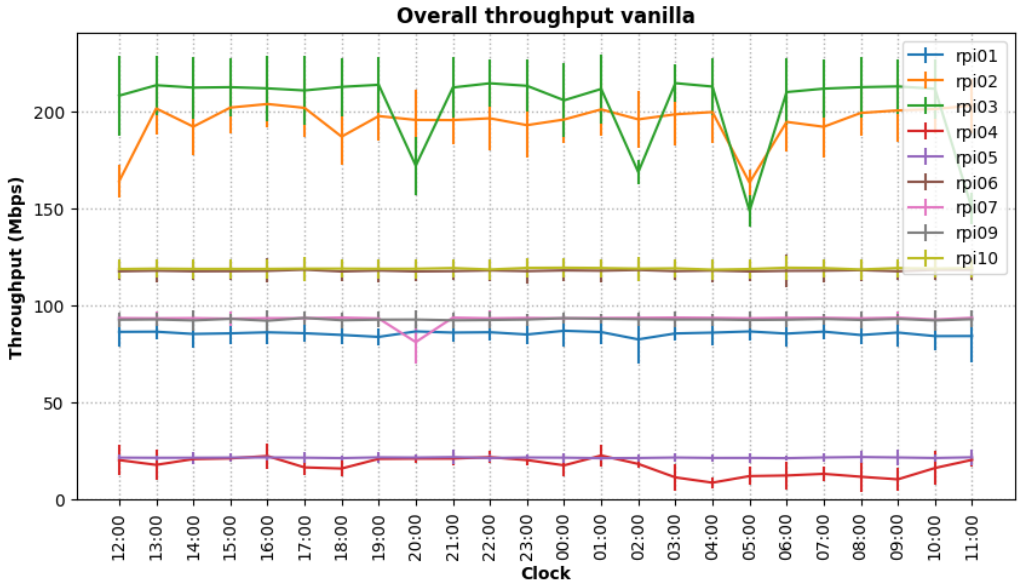


Figure 6.5: Weekly raw throughput as experienced by all nodes.

6.3.1 Raw Throughput

We used our RPi deployment to run periodic *iperf3* throughput experiments to measure the throughput without the usage of Moby. Every RPi device would, hourly, connect to a remote server on Google Cloud and measure its throughput. We utilized a different port, on the same machine, for the 9 RPi in our deployment. To determine the throughput we had the iperf3 run 30 iterations of measurements which we used to determine the median and standard deviation.

Figure 6.5 displays our throughput results for Moby-less *iperf3* testing spanning the course of 24 hours, starting noon and ending noon the day after. As we cab observe the 9 RPi offer greatly differing bandwidths between devices, and some variability in the sustained throughput over the course of a day. Not only this figure shows the heterogeneity of our environment – considering the different ISPs, Modems, and connection types – but also allows to observe that the most high-performing devices are also the most volatile when it comes to throughput variability.

6.3.2 Raw Latency

The same logic used for throughput measurement is also applied to the raw latency evaluation for our deployment of the RPi devices. For this experiment, every RPi would, once every 2 minutes, use *httping* on the same Google Cloud machine used for the previous experiments. To determine the latency we had *httping* run 30 iterations of measurements allowing us to determine the median and standard deviation.

Figure 6.6 displays our latency results for Moby-less *httping* testing spanning the course of 24 hours, starting noon and ending noon the day after. As we can observe, the latency for every RPi was about the same, ranging from 40 to 50 ms. This is explained by the fact that every device is located around the metropolitan area of Lisbon, and our Google Cloud testing server is located in Frankfurt.

Since the distance between every device located in Lisbon to Frankfurt is roughly the same (≈2315

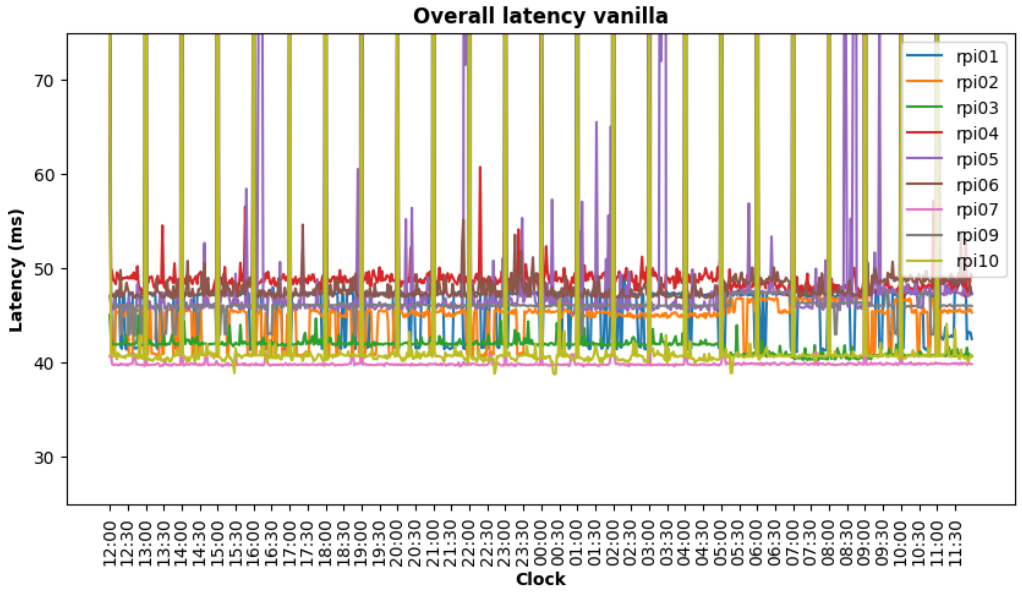


Figure 6.6: Weekly raw latency as experienced by all nodes.

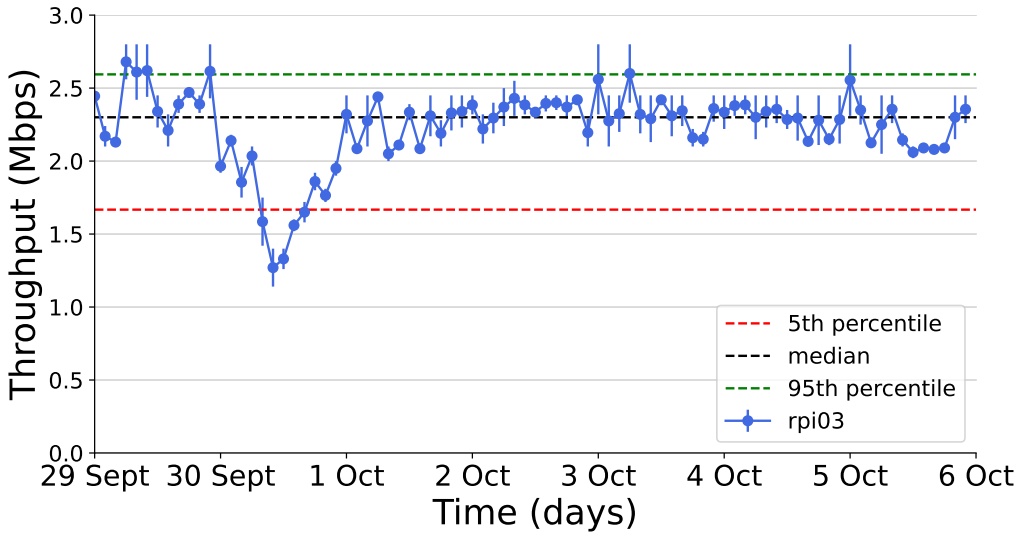
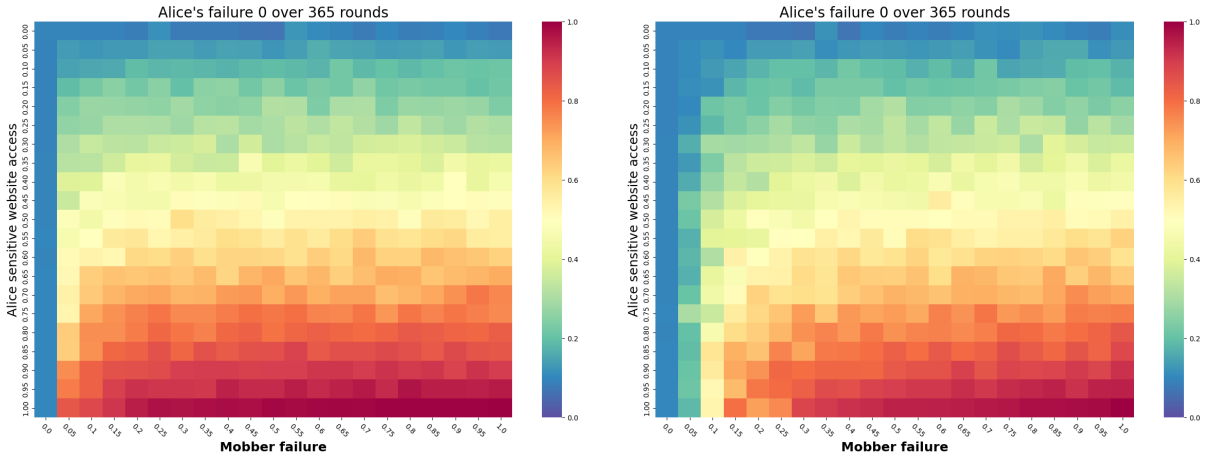


Figure 6.7: Weekly Moby throughput as experienced by *rpi3*.

km) the interesting observations would be the latency spikes observed on *rpi05* and *rpi10*. These observed spikes, reaching out the frame, are mostly due to temporary failures of connection. The users of this study were queried about the nature of these failures with the prevalent answer being faulty network service. During the duration of this study no power outages were observed thus allowing for a high availability observation across our 9 RPis.

6.3.3 Flashmob Throughput

Following the previous raw evaluation, we now run the same testing procedures for throughput but using Moby flashmobs. We used our RPi deployment to run periodic Moby rounds based on a k-circuit (k=9) while using a traffic shaping rate of ≈5 Mbps, i.e., equivalent to 720p video streaming. Every two hours, the RPis engaged in a Moby round where device *rpi3* sent real data, in the form of an `iperf3` measurement, and all other nodes sent chaff. In this experiment, all RPi nodes were connected via the



(a) Standard distribution with 100% chance of failures being permanent

(b) Standard distribution with 25% chance of failures being permanent

Figure 6.8: Original statistical disclosure for a generic scenario

same Tor circuit. We selected the nodes composing this circuit by picking a Tor middle relay and exit relay with an uptime larger than 40 days and a total advertised throughput over 20 Mbps.

Figure 6.7 shows our weekly throughput measurements conducted over Moby. The figure reveals that Moby's traffic was rather stable, achieving a median throughput of 2.3 Mbps. Some of the more meaningful throughput drops we were able to observe (e.g., in September 30th), may be explained by the fact that the operator of *rpi3* was working from home and attending back-to-back videoconferencing meetings throughout the day, using another machine. This suggests that the congestion caused by other bandwidth-hungry applications' traffic in the network adds to the variability of Moby's performance.

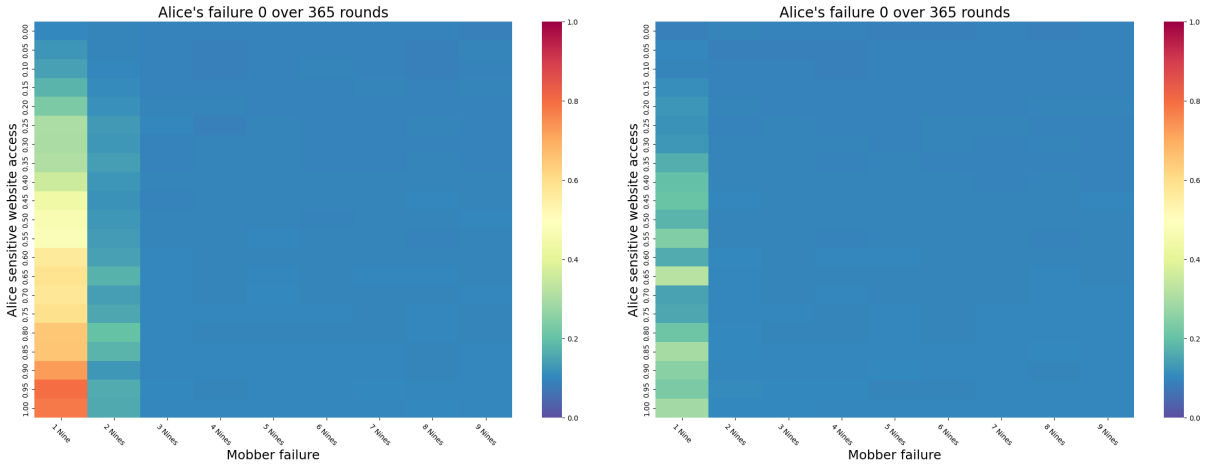
6.4 Resistance Against Statistical Disclosure

One of the main components of our work is to evaluate flashmobs composed within Moby against possible statistical disclosure on the anonymity sets. The analysis of this topic will follow these steps: (i) first, we theorize how the original statistical disclosure attack and the improved generalized version affects the profile of Alice; (ii) second, we evaluate the performance of flashmobs under a high availability scenario, (iii) third, we observe how the omission of Alice from a flashmobs makes it more difficult for an attacker to profile Alice's website accesses; and (iv) fourth, it is inferred how the usage of an oracle on Moby bridges with a minimum threshold for the anonymity sets would influence Alice's anonymity.

6.4.1 Expected Results

To measure Moby's resistance to statistical disclosure attacks we implemented the original attack introduced by Danezis in 2003 [56]. In this section, we set the stage for the scenarios presented in the next sections and more closely resembling possible real-world scenarios such as the setting presented in Section 6.3.

Figure 6.8 shows an heatmap for the original statistical disclosure probability attack against Alice when profiling her website accessing under the guise of a flashmob. We consider a total of 100 mobbers



(a) Highly available users with 100% chance of failures being permanent

(b) Highly available users with 25% chance of failures being permanent

Figure 6.9: Original statistical disclosure for highly available users

– $\mathbb{K}$ – and 100 websites – $\mathbb{M}$ – any member of the flashmob, can access to. Although for statistical disclosure, one takes the entire vector of accessed websites, for these heatmaps we only display the probability of accessing a single sensitive website – WikiLeaks. We assume the mobbers in a flashmob each have the same probability of accessing each of the websites in $\mathbb{M}$ in any of the communication rounds. As such, the naive approach for determining the probability of a single user accessing any of the websites in the set would be $\frac{1}{\#\mathbb{K}}$.

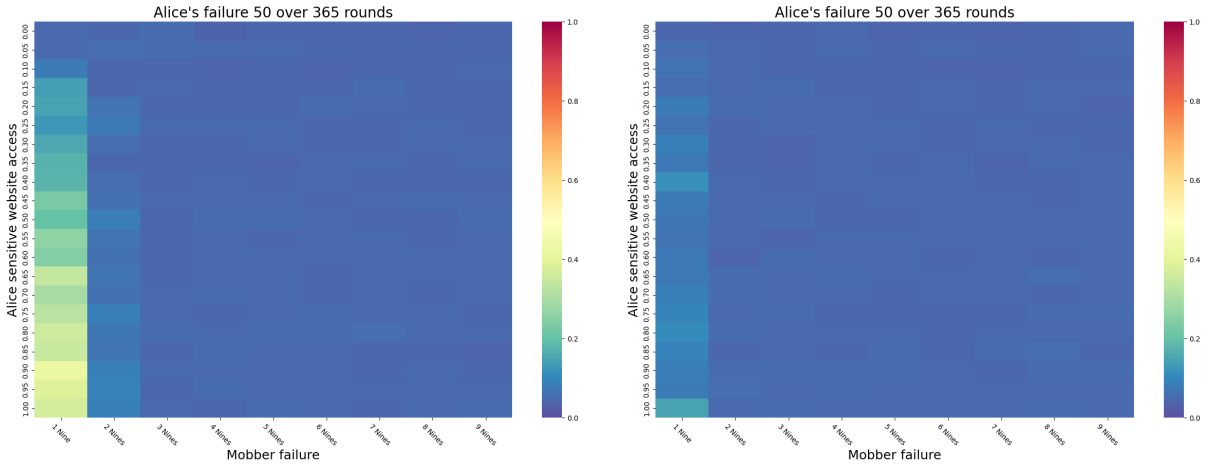
This heatmap was generated following 365 rounds (assuming a flashmobs scheduled daily for one year). On the *x* axis we present the failure rate of the peers of Alice in a flashmob. We vary this rate from 0% (never failing) to 100% (failing every round). On the *y* axis we present the probability of Alice accessing the sensitive website more often than her peers. Assuming her peers have $\frac{1}{\#\mathbb{M}}$ probability of accessing any website in the list, Alice has x probability of accessing WikiLeaks and $\frac{1-x}{\#\mathbb{M}-1}$ probability of accessing any of the remaining websites.

For every heatmap the bar on right-side presents the observed probability of the attacker compared with Alice's real probability of WikiLeaks. What we can observe is that, assuming a failure results in permanent churn 6.8(a), with no defense mechanism deployed, the observed probability by the attacker would, after 365 rounds, match the real probability of Alice's WikiLeaks accesses on column *y*.

Furthermore, in the following heatmaps we evaluate the variation of the probability of failures resulting in permanent churn, i.e., if in 6.8(a) we assumed the churn for 100 users was permanent (100% probability of no user returning upon failure), in 6.8(b) the probability of not returning when failing decreases to 25% and hence it greatly decreases the profiling of Alice. What essentially this shows us is that as long as we have a certain threshold in the minimal amount of users showing up to a flashmob it will greatly hamper the adversary's perception of Alice's WikiLeaks access probability.

6.4.2 High Availability Scenario

Following our evaluation for a theoretical availability of flashmob users, which includes hypothesized lower-availability scenarios, we now evaluate the flashmob composition for the opposite – high availabil-



(a) Highly available users with 100% chance of failures being permanent

(b) Highly available users with 25% chance of failures being permanent

Figure 6.10: Original statistical disclosure for Alice's 50% absence rate with high-available mobbers

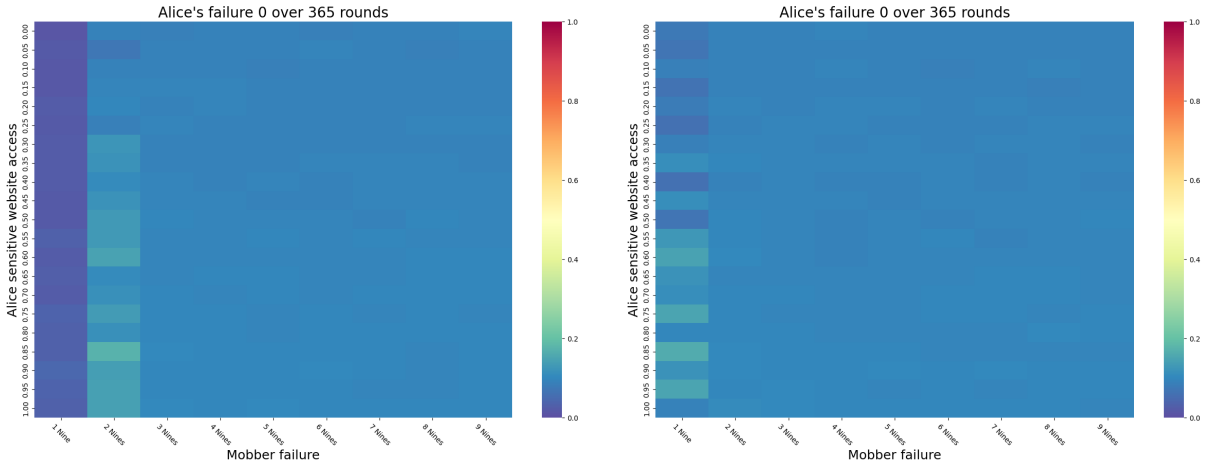
ity. In Section 6.3 we demonstrate that real-world deployments of Moby are capable of high availability, thus providing strong protection against statistical disclosure.

Figure 6.9 shows a highly available scenario for flashmob compositions. Although similar to the heatmaps shown in the previous Section 6.4.1 on figure 6.8, the main difference is that for the x axis, we vary the failure rate of mobbers using the number of nines [95]. Despite a different variation of the x axis, the number of $\#\mathbb{K}$ continues to be 100, and the number of $\#\mathbb{M}$ stays also at 100.

As we can observe, in Figure 6.9(a) for a failure rate of 10% (1 Nine) and with permanent churn of every user (100% probability of not return upon failure), assuming Alice accesses WikiLeaks 100% of the time, the attacker infered probability is $\approx$70%. As such, the attacker infers with a $\approx$30% the difference of the WikiLeaks accessing profiling of Alice. However, even if every failure means permanent churn in the flashmob, for (2 Nines) the attacker can barely infer the probability of Alice accessing the sensitive website being monitored.

For Figure 6.9(b) for a failure rate of 10% (1 Nine) and with 25% probability of a mobber failure meaning permanent churn, the attacker has a great difficulty of profiling Alice's probability of accessing WikiLeaks. Even when Alice accesses WikiLeaks, due to high availability of mobbers and low permanent churn probability, for the scenario where Alice accesses WikiLeaks 100% of the time, the attacker infers the probability of Alice as $\approx$35%. As such, the difference between the real probability and the observed by the attacker is $\approx$75%.

As such, we can conclude that highly available scenarios can concur in great protection for flashmob participants as long as mobber failure probability is no more than 10%. Furthermore, as shown, not only does the failure probability of mobbers influence the attacker readings, but the analysis for temporary versus permanent churn, has shown us that, the lower the probability of permanent churn, the better Alice is protected against a global passive adversary.



(a) Highly available users with 100% chance of failures being permanent

(b) Highly available users with 100% chance of failures being permanent

Figure 6.11: Original statistical disclosure for high-available mobbers with Oracle

6.4.3 Absences in Flashmobs

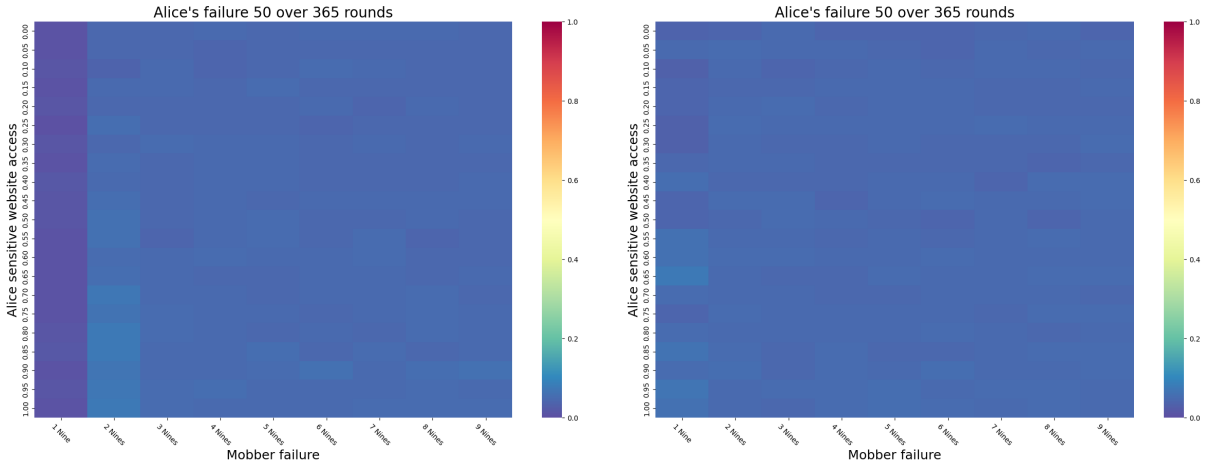
From the observations shown in the previous sections, we concluded that for flashmobs to be effective three main factors must be balanced: (i) the number of clients in a flashmob, (ii) the availability of mobbers, and (iii) the probability of failures being temporary. Taking these factors into account the next step of our study evaluates how Alice absenting herself from some communication rounds will affect the profiling of Alice by the attacker.

Figure 6.10 evaluates how Alice being offline 50% of the scheduled rounds affects her profiling. The parameters used are the same as in previous rounds, the only difference being that Alice will be offline for half of the daily scheduled rounds. Figure 6.10(a) shows Alice's profiling when churn is permanent with her flashmob peers. Comparing the observed probability with Alice's real probability of accessing a website in $\mathbb{K}$, when Alice accesses WikiLeaks 100% of the time, the attacker infers a probability of $\approx$40%. As such, we can observe that this approach lowers the attackers inference in about $\approx$60%. In Figure 6.10(b) we observe that when capping the probability of failures being permanent at 25%, Alice being offline for 50% of the time and accessing WikiLeaks every round she is online, the attacker perception is that Alice accesses WikiLeaks $\approx$20% of the time. As such the difference between the calculated and the actual probability is $\approx$80%.

6.4.4 Oracle and Minimum Threshold

We can observe from previous experiments that the amount of offline mobbers not only conditions the plausible deniability of Alice, but also for greater probabilities of permanent churn, the attacker is able to greatly approximate the real accesses profile with the estimated one. As such, and inspired by Wolinksy [22], when traffic is routed through a Moby bridge, this component can access how many messages it did receive and implement a minimum threshold of online mobbers – $\mathbb{O}$ – to prevent the deanonymization of users, thus acting like an Oracle.

In Figure 6.11 we implement a minimum flashmob threshold of $\mathbb{O} = 4$ for a flashmob. If during a certain communication round the Moby detects that no more than 4 messages were sent to the destination



(a) SDA with failure rate of 10% and max. of 1 message per round.

(b) ISDA with failure rate of 1% and max. of 100 messages per round.

Figure 6.12: Numero de noves para 100 gajos com churn permanente variavel.

websites no traffic will be routed. Following the evaluation on previous sections, the parameters remain the same for the $\mathbb{K}$, websites being accessed $\mathbb{M}$ and probability of a failure resulting in permanent churn.

In Figure 6.11(a) we can observe that for a failure rate of 10% (1 nine), with every failure resulting in permanent churn, regardless of Alice's sensitive website accessing tendencies, barely any message is being routed through Moby. Although the attacker cannot profile Alice, this poses an usability issue. Nonetheless, for a mobber failure rate of 1% (2 Nines) the results are much more encouraging. Not only, does the attacker have trouble profiling Alice's WikiLeaks accesses (compared to her real probability) but rounds are not being blocked by the Moby bridge Oracle. For , assuming that Alice accesses WikiLeaks 100% of the time, the attacker perceives Alice's probability of accessing WikiLeaks as ≈5%. The difference between the actual probability and the perceived one is ≈95%.

On the other hand, Figure 6.11(b) shows that for a failure rate of 10% (1 nine) with only 25% of the failures resulting in permanent churn, the attacker cannot differentiate the instance where Alice accesses WikiLeaks from the naive scenario where her probability would be $\frac{1}{\mathbb{M}}$. What is meant by this, is that for any of the *x* axis parameters, for any WikiLeaks access probability by Alice's, the attacker perceives the probability of accessing this website as ≈1%, the same probability as accessing any of the $\mathbb{M}$ websites at random.

As such, we can conclude that for Moby flashmobs to provide usability a degree of usability, the flashmob users have to possess highly availability (above 90% per round) and a low probability of failures resulting in permanent churn. With these conditions known, we can greatly vary the size of the flashmob sizes and the $\mathbb{O}$ online mobber threshold. We leave for future work evaluating the correlation between flashmob sizes, availability and the minimum flashmob threshold necessary to route traffic between bridges.

Finally, we present on Figure 6.12 a corner case of the Moby bridge oracle implementation. As shown in Section 6.4.3, the possibility of Alice absenting herself from the flashmob performances, when the Oracle is implemented, makes it more difficult for the attacker to properly profile Alice. Through the combination of Alice being offline 50% of her rounds with Moby bridge oracle requiring $\mathbb{O} \geq 4$ to route traffic, figure 6.12(a) shows us that for 10% failure of mobbers and permanent churn, Alice cannot be

profiled. This mostly happens because of a combination of: (i) Alice being offline, (ii) not enough users being able to constitute a flashmob of 4 users after 365 rounds.

Since the above requirements are very strict and make Moby performance falter, figure 6.12(b) assumes a more lenient parameter of instead 25% of failures resulting in permanent churn. We observe the attacker is unable to distinguish Alice's accessing profile from a random distribution of website accesses, even for the case where every available round Alice accesses WikiLeaks.

Summary

This chapter presented our experimental evaluation of Moby. It showed that Moby is able to provide a reasonable throughput and latency needed for typical network activities (e.g. browsing or video streaming) even when behind residential NATs. Moby has shown to be resilient against statistical disclosure specifically in the presence of a state-level adversary analysing traffic flows and correlating with anonymity sets. However, depending on the configuration parameters, specifically the anonymity set size, and the threshold to avoid deanonymization, usability might be compromised. The next chapter concludes this document by reviewing the key points of this thesis and introduce the future work.

Chapter 7

Conclusions

Nowadays, attacking anonymity networks is becoming easier for state-level adversaries. Although recent works such as TorK add k-anonymity and prevent traffic flow correlation on Tor (the most popular anonymity network in usage as the time of writing), such system remains vulnerable to intersection attacks and statistical disclosure against a global passive adversary. In this work, we presented Moby, aimed at providing Tor a network of Moby bridges protecting whistleblowers from intersection attacks and statistical disclosure. This is achieved by introducing k-anonymous flashmobs.

7.1 Achievements

With this work our major achievements were the design and implementation of Moby, a Tor pluggable transport and controller which offers additional protection against intersection attacks and statistical disclosure. By leveraging TorK and scheduling a *k*-anonymous flashmob whistleblowers are provided plausible deniability since an attacker is not able to correlate traffic flows during the period the anonymity set is active. Furthermore, we have deployed Moby across a major European city metropolitan area using IoT devices in order to study how a high availability scenario would perform against a global-passive adversary. Finally, for intersection attacks and statistical disclosure we have studied how Moby would resist against these types of attacks and what are the ideal features to strengthen flashmobs, mostly: (i) the size of anonymity sets, (ii) the probabilities of mobber churn, (iii) the permanent user churn, (iv) and the usage of an oracle implementing a minimum user threshold to mitigate disclosure.

7.2 Future Work

Our work on Moby allows us to identify multiple interesting directions for future work. First, the implementation of Moby should support a global mesh of bridges in order for flashmobs to be distributed across different relays and balance the induced load. Second, we an interesting point to study would be to infer how to attribute specific browsing patterns to mobbers instead of just having these retrieve webpages or emitting chaff. A third challenge would be the recruiting of clients for flashmobs. Currently

we have utilized a model where the flashmob creator knows which are the clients that she wants to recruit. Furthermore, no incentive is given to mobbers in order for them to participate in anonymity sets. For future due to participation in flashmobs, the usage of smart contracts, cryptocurrency, or even reputation gains could be used to reward mobbers for their participation. A fourth interesting challenge would be the advertising of flashmobs for users in the real world. As such, one could explore the possibilities of having clients join flashmobs based on the anonymity guarantees that could be given to the user. These parameters could for example be, the advertise minimum number of users for a flashmob to occur, the average number of users and churn rate of past flashmobs, and the dates and durations for the anonymity set rounds, and the types of websites one is able to access during flashmob performances. To this end, it is important for mobbers to query bridges for the characteristics of the currently stored flashmobs. A fifth and final challenge would be to analyse the probabilities of users in Moby using the generalized statistical disclosure attack and the least squares statistical disclosure attack, to determine the correct parameters

Bibliography

[1] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium, August 9-13, San Diego, California, USA*, 2004.

[2] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communications*, 1998.

[3] Tor Project. Tor FAQ. `https://2019.www.torproject.org/about/overview.html.en`, . Accessed: 2021-09-10.

[4] J. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, California, USA, July 25-26*, 2000.

[5] S. J. Murdoch and G. Danezis. Low-Cost Traffic Analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (S&P 2005), 8-11 May, Oakland, California, USA*, 2005.

[6] J. Barker, P. Hannay, and P. Szewczyk. Using Traffic Analysis to Identify the Second Generation Onion Router. In *Proceedings of the 9th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Melbourne, Australia, October 24-26*, 2011.

[7] Tor Project. Traffic Correlation Using Netflows. `https://blog.torproject.org/traffic-correlation-using-netflows?page=1`, . Accessed: 2021-10-29.

[8] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, Berlin, Germany, November 4-8*, 2013.

[9] K. Gallagher. How Tor helped catch the Harvard bomb threat suspect. `https://www.dailydot.com/crime/tor-harvard-bomb-suspect/`, 2016. Accessed : 2021-11-02.

[10] R. Lakshmanan. Russia Blocks Tor Privacy Service in Latest Censorship Move. `https://amp.thehackernews.com/thn/2021/12/russia-blocks-tor-privacy-service-in.html`. Accessed: 2021-12-12.

[11] Wired. Russia's Internet Censorship Machine Is Going After Tor. `https://www.wired.com/story/russia-block-tor-censorship/amp`. Accessed: 2021-12-12.

[12] D. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable Anonymous Group Communication in the Anytrust Model. In *Proceedings of the 5th ACM European Workshop on Systems Security, April 10*, 2012.

[13] H. Corrigan-Gibbs and B. Ford. Dissent: Accountable Anonymous Group Messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, Chicago, Illinois, USA, October 4-8*, 2010.

[14] L. Barman, I. Dacosta, M. Zamani, E. Zhai, A. Pyrgelis, B. Ford, J. Feigenbaum, and J. Hubaux. PriFi: Low-Latency Anonymity for Organizational Networks. *Proceedings on Privacy Enhancing Technologies*, 2020.

[15] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso. TARANET: Traffic-Analysis Resistant Anonymity at the Network Layer. In *Proceedings of the 2018 IEEE European Symposium on Security and Privacy, London, United Kingdom, April 24-26*, 2018.

[16] A. Kwon, D. Lu, and S. Devadas. XRD: Scalable Messaging System with Cryptographic Privacy. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation, Santa Clara, California, USA, February 25-27*, 2020.

[17] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle: An Efficient Communication System With Strong Anonymity. *Proceedings of the Privacy Enhancing Technologies*, 2016.

[18] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. F. Syverson. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, Berlin, Germany, November 4-8*, 2013.

[19] V. Nunes. Hardening Tor against State-Level Traffic Correlation Attacks with K-Anonymous Circuits. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, 2021.

[20] G. Danezis and A. Serjantov. Statistical Disclosure or Intersection Attacks on Anonymity Systems. In *Proceedings of the 6th International Conference on Information Hiding, Toronto, Canada, May 23-25*, 2004.

[21] S. Oya, C. Troncoso, and F. Pérez-González. Meet the Family of Statistical Disclosure Attacks. *Computing Research Repository*, 2019.

[22] D. I. Wolinsky, E. Syta, and B. Ford. Hang With Your Buddies to Resist Intersection Attacks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, Berlin, Germany, November 4-8*, 2013.

[23] J. Hayes, C. Troncoso, and G. Danezis. TASP: Towards Anonymity Sets that Persist. In *Proceedings of the 2016 ACM Workshop on Privacy in the Electronic Society, Vienna, Austria, October 24-28*, 2016.

[24] E. G. Sirer, S. Goel, M. Robson, and D. Engin. Eluding Carnivores: File Sharing With Strong Anonymity. In *Proceedings of the 11st ACM SIGOPS European Workshop, Leuven, Belgium, September 19-22*, 2004.

[25] L. Melis, G. Danezis, and E. D. Cristofaro. Efficient Private Statistics with Succinct Sketches. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium, San Diego, California, USA, February 21-24*, 2016.

[26] D. Lazar and N. Zeldovich. Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, Georgia, USA, November 2-4*, 2016.

[27] N. Gelernter, A. Herzberg, and H. Leibowitz. Two Cents for Strong Anonymity: The Anonymous Post-Office Protocol. In *Proceedings of the 16th International Conference on Cryptology and Network Security, Hong Kong, China, November 30-December 2*, 2017.

[28] S. Goel, M. Robson, M. Polte, and E. Sirer. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. *Technical Report, Cornell University*, 2003.

[29] S. L. Blond, D. R. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, London, United Kingdom, August 17-21*, 2015.

[30] M. J. Freedman and R. T. Morris. Tarzan: A Peer-to-peer Anonymizing Network Layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington DC, USA, November 18-22*, 2002.

[31] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An Anonymous Messaging System Handling Millions of Users. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, California, USA, May 17-21*, 2015.

[32] S. Matic, C. Troncoso, and J. Caballero. Dissecting Tor Bridges: A Security Evaluation of their Private and Public Infrastructures. In *Proceedings of the 2017 Network and Distributed System Security Symposium, San Diego, California February 26-March 1*, 2017.

[33] Internet Engineering Task Force. RFC 6066: Transport Layer Security (TLS) Extensions: Extension Definitions. `https://tools.ietf.org/html/rfc6066`. Accessed: 2022-01-05.

[34] Tor Project. A child garden of pluggable transports. `https://trac.torproject.org/projects/tor/wiki/doc/AChildsGardenOfPluggableTransports`, . Accessed: 2022-01-05.

[35] Tor Project. Tor Pluggable Transports. `https://2019.www.torproject.org/docs/pluggable-transports`, . Accessed: 2022-01-05.

[36] Tor Project. meek. `https://trac.torproject.org/projects/tor/wiki/doc/meek`, . Accessed: 2022-01-05.

[37] Yawning Angel. obfs4 (The obfourscator). https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt. Accessed: 2022-01-05.

[38] Tor Project. obfs2 (the twobfuscator). https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs2/obfs2-protocol-spec.txt, . Accessed: 2022-01-05.

[39] S. Chakravarty, A. Stavrou, and A. D. Keromytis. Traffic Analysis Against Low-Latency Anonymity Networks Using Available Bandwidth Estimation. In *Proceedings of the 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22*, 2010.

[40] R. Pries, W. Yu, X. Fu, and W. Zhao. A New Replay Attack Against Anonymous Communication Networks. In *Proceedings of the 2008 IEEE International Conference on Communications, Beijing, China, May 19-23*, 2008.

[41] S. J. Murdoch and G. Danezis. Low-cost Traffic Analysis of Tor. In *Proceedings in the 2005 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-11*, 2005.

[42] B. Evers, J. Hols, E. Kula, J. Schouten, M. den Toom, R. van der Laan, and J. Pouwelse. Thirteen Years of Tor Attacks. *Technical Report, Delft University of Technology*.

[43] P. Winter, R. Ensafi, K. Loesing, and N. Feamster. Identifying and Characterizing Sybils in the Tor Network. In *Proceedings of the 25th USENIX Security Symposium, Austin, Texas, USA, August 10-12*, 2016.

[44] J. R. Douceur. The Sybil Attack. In *Proceedings of the 2002 First International Workshop on Peer-to-Peer Systems, Cambridge, Massachusetts, USA, March 7-8*, 2002.

[45] K. Bauer, D. McCoy, D. C. Grunwald, and T. Kohno. Low-Resource Routing Attacks Against Anonymous Systems. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society, New York, New York, USA, October 29*, 2007.

[46] D. R. Figueiredo, P. Nain, and D. Towsley. On the Analysis of the Predecessor Attack on Anonymity Systems. *Computer Science Technical Report*, 2004.

[47] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending Anonymous Communications Against Passive Logging Attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 11-14*, 2003.

[48] R. Dingledine and G. Kadianakis. One Fast Guard for Life (or 9 months). In *Proceedings of the 7th Workshop on Hot Topics in Privacy Enhancing Technologies, Amsterdam, Netherlands*, 2014.

[49] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. RAPTOR: Routing Attacks on Privacy in Tor. In *Proceedings in the 24th USENIX Security Symposium, Washington DC, USA, August 12-14*, 2015.

[50] V. Shmatikov and M. Wang. Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses. In *Proceedings of the 11th European Symposium on Research on Computer Security, Hamburg, Germany, September 18-20*, 2006.

[51] M. Nasr, A. Bahramali, and A. Houmansadr. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, Ontario, Canada, October 15-19*, 2018.

[52] R. Nithyanand, O. Starov, P. Gill, A. Zair, and M. Schapira. Measuring and Mitigating AS-level Adversaries Against Tor. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium, San Diego, California, USA, February 21-24*, 2016.

[53] M. Akhoondi, C. Yu, and H. V. Madhyastha. LASTor: A Low-Latency AS-Aware Tor Client. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy, San Francisco, California, USA, May 21-23*, 2012.

[54] M. Edman and P. F. Syverson. AS-awareness in Tor path selection. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security, Chicago, Illinois, USA, November 9-13*, 2009.

[55] S. E. Oh, T. Yang, N. Mathews, J. Holland, M. Rahman, N. Hopper, and M. Wright. DeepCoFFEA: Improved flow correlation attacks on tor via metric learning and amplification. In *Proceedings of the 2022 IEEE Symposium on Security and Privacy*, 2022.

[56] G. Danezis. Statistical Disclosure Attacks. In *Proceedings of the International Conference on Information Security on Security and Privacy in the Age of Uncertainty, Athens, Greece, May 26-28*, 2003.

[57] N. Mathewson and R. Dingledine. Practical Traffic Analysis: Extending and Resisting Statistical Disclosure. In *Proceedings of the 4th International Workshop on Privacy Enhancing Technologies, Toronto, Canada, May 26-28*, 2004.

[58] A. Serjantov, R. Dingledine, and P. F. Syverson. From a Trickle to a Flood: Active Attacks on Several Mix Types. In *Proceedings of the 5th International Workshop on Information Hiding, Noordwijkerhout, The Netherlands, October 7-9*, 2002.

[59] F. Pérez-González and C. Troncoso. Understanding Statistical Disclosure: A Least Squares Approach. In *Proceedings of the 12th International Symposium on Privacy Enhancing Technologies, Vigo, Spain, July 11-13*, 2012.

[60] Tor Project. Tor FAQ. `https://2019.www.torproject.org/docs/faq.html.en`, . Accessed: 2022-01-05.

[61] Tor Project. Reporting Bad Relays. `https://trac.torproject.org/projects/tor/wiki/doc/ReportingBadRelays`, 2018. Accessed: 2022-01-05.

[62] Tor Project. How to Report Bad Relays. https://blog.torproject.org/how-report-bad-relays, 2014. Accessed: 2022-01-05.

[63] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. R. Weippl. Spoiled Onions: Exposing Malicious Tor Exit Relays. In *Proceedings of the 14th International Symposium on Privacy Enhancing Technologies, Amsterdam, Netherlands, July 16-18*, 2014.

[64] S. Chakravarty, G. Portokalidis, M. Polychronakis, and A. D. Keromytis. Detecting traffic snooping in tor using decoys. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection, Menlo Park, California, USA, September 20-21*, 2011.

[65] T. Elahi, K. S. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg. Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor. In *Proceedings of the 11th annual ACM Workshop on Privacy in the Electronic Society, Raleigh, North Carolina, USA, October 15*, 2012.

[66] J. Juen, A. Johnson, A. Das, N. Borisov, and M. Caesar. Defending Tor from Network Adversaries: A Case Study of Network Path Prediction. *Proceedings on Privacy Enhancing Technologies*, 2015.

[67] C. Wacek, H. Tan, K. S. Bauer, and M. Sherr. An Empirical Evaluation of Relay Selection in Tor. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium, San Diego, California, USA, February 24-27*, 2013.

[68] Z. Li, S. Herwig, and D. Levin. DeTor: Provably Avoiding Geographic Regions in Tor. In *Proceedings of the 26th USENIX Security Symposium, Vancouver, British Columbia, Canada, August 16-18*, 2017.

[69] D. Levin, Y. Lee, L. Valenta, Z. Li, V. Lai, C. Lumezanu, N. Spring, and B. Bhattacharjee. Alibi Routing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, London, United Kingdom, August 17-21*, 2015.

[70] Z. Weinberg, S. Cho, N. Christin, V. Sekar, and P. Gill. How to Catch when Proxies Lie: Verifying the Physical Locations of Network Proxies with Active Geolocation. In *Proceedings of the 2018 Internet Measurement Conference, Boston, Massachusetts, USA, October 31-November 2*, 2018.

[71] K. Kohls, K. Jansen, D. Rupprecht, T. Holz, and C. Pöpper. On the Challenges of Geographical Avoidance for Tor. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium, San Diego, California, USA, February 24-27*, 2019.

[72] D. Das, S. Meiser, E. Mohammadi, and A. Kate. Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency - Choose Two. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy, San Francisco, California, USA*, 2018.

[73] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 11-14*, 2003.

[74] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in Numbers: Making Strong Anonymity Scale. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation, Hollywood, California, USA, October 8-10*, 2012.

[75] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford. Proactively Accountable Anonymous Messaging in Verdict. In *Proceedings of the 22th USENIX Security Symposium, Washington DC, USA, August 14-16*, 2013.

[76] P. Samarati and L. Sweeney. Generalizing Data to Provide Anonymity when Disclosing Information. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Seattle, Washington, USA, June 1-3*, 1998.

[77] A. Biryukov and I. Pustogarov. Proof-of-Work as Anonymous Micropayment: Rewarding a Tor Relay. In *Proceedings of the 19th International Conference on Financial Cryptography and Data Security, San Juan, Puerto Rico, January 26-30*, 2015.

[78] T. Dinh, F. Rochet, O. Pereira, and D. S. Wallach. Scaling Up Anonymous Communication with Efficient Nanopayment Channels. *Proceedings on Privacy Enhancing Technologies*, 2020.

[79] E. Eşanu. Gamification: Understanding the basics. `https://uxplanet.org/gamification-understanding-the-basics-2bbcce365c33`. Accessed: 2022-01-12.

[80] Level Skip. The Endless Grind: "World of Warcraft" Reputation Guide. `https://levelskip.com/mmorpgs/World-of-Warcraft-Reputation-Guide`. Accessed: 2022-01-12.

[81] E. Snowden. Tor Stinks. `https://edwardsnowden.com/docs/doc/tor-stinks-presentation.pdf`. Accessed: 2021-11-04.

[82] D. N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-Resilient Online Content Voting. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, Boston, Massachusetts, USA, April 22-24*, 2009.

[83] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A Near-Optimal Social Network Defense Against Sybil Attacks. *IEEE/ACM Transactions on Networking*, 2010.

[84] A. H. Lorimer, L. Tulloch, C. Bocovich, and I. Goldberg. OUStralopithecus: Overt User Simulation for Censorship Circumvention. In *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society, Virtual Event, Korea, November 15*, 2021.

[85] V. Costan and S. Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016.

[86] M. Leech, M. Ganis, Y.-D. Lee, R. Kuris, D. Koblas, and L. Jones. Rfc 1928: Socks protocol version 5. `https://tools.ietf.org/html/rfc1928`, 1996. Accessed: 2022-01-05.

[87] Tor Project. Tor control protocol. `https://gitweb.torproject.org/torspec.git/tree/control-spec.txt`, Jul. 2020.

[88] N. Lomas. Waiting for raspberry pi: Eben upton talks supply constraints and demand shock. https://techcrunch.com/2022/10/05/raspberry-pi-supply-chain-shocks-eben-upton-interview/?guccounte=1. Accessed: 2022-10-30.

[89] Google. Google cloud: Cloud computing services. https://cloud.google.com/. Accessed: 2022-10-15.

[90] Tailscale Inc. Tailscale - best vpn service for secure networks. https://tailscale.com/. Accessed: 2022-10-25.

[91] Ansible Inc. / Red Hat Inc. Ansible is simple it automation. https://www.ansible.com/. Accessed: 2022-10-25.

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

[93] The Tor Project. Network ddos. https://status.torproject.org/issues/2022-06-09-network-ddos/. Accessed: 2022-10-31.

[94] Background_Ad6358. Tor ddos attacks. https://www.reddit.com/r/TOR/comments/vkwib2/tor_ddos_attacks/. Accessed: 2022-10-31.

[95] C. of Wikipedia. High availability. https://en.wikipedia.org/wiki/High_availability. Accessed: 2022-10-29.