Privacy Protection against Passive Monitoring using SDNs

by

Pushkar Bettadpur

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Privacy Protection against Passive Monitoring using SDNs

Pushkar Bettadpur
Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2019

In traditional networks, an adversary who compromises one or more forwarding elements (switches, links, routers) automatically gains the ability to passively monitor a large amount of traffic. Even if traffic payloads are encrypted, the adversary can infer a lot from observing routing (network layer) headers, such as sender-receiver pairs, the amount of data sent, and confidential data within encrypted packets. To address this, we present TrafFu, a system that protects network layer headers from passive monitoring using software defined networks (SDN). TrafFu encrypts end-host identifiers in packet headers namely, the IP addresses, at the source. Forwarding elements rely on a trusted controller to decrypt the packets and provide forwarding rules. By using different keys to encrypt packets in a flow, TrafFu breaks the mapping between a packet and its network flow, making traffic analysis attacks more difficult. TrafFu's main cost is that it increases flow table utilization on forwarding elements, which we measure in our evaluation. We also propose a method to allow some aggregation of flows to reduce flow table pressure at the cost of some loss of privacy, and study this trade-off.

Acknowledgements

I would like to express my heartfelt gratitude to my thesis supervisor, Professor David Lie, who was a constant source of help, knowledge and guidance throughout the duration of this project, and my term as a student on the whole.

I would also like to thank my fellow contributors to this project namely, He (Shawn) Shuang, Wei Huang, Ivan Pustogarov and Tao Wan. Their insights, opinions and feedback were the fundamental constructs to this project.

I am extremely thankful to the University of Toronto and the Department of Electrical and Computer Engineering for giving me a platform to learn and explore the boundaries of Science and Technology for over 7 years, as both an Undergraduate and Graduate student. As a recipient of the Ontario Graduate Scholarship, I am very grateful for the financial support received by the University and the Government of Ontario.

Lastly but most importantly, I would like to thank my family and friends for their constant support over the years. Their permanence in presence made my life as easy as it could get.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

One of the biggest challenges facing society today is the security of web traffic. The ubiquity of activities conducted online makes network traffic a resource of considerable interest for a number of parties. One manifestation of this interest is in the form of covert surveillance, wherein an attacker looks to gain information about the users of a network while remaining undetected. Instances of such *attacks* have risen to the degree that espionage motivated cyber attacks are now second only to profit-motivated cybercrime as a reason for attackers to hack into networks [57]. Typically, such attacks bring to mind the compromise of internal system machines. However, such machines are often hardened, updated with the latest security patches and tend to be closely monitored, increasing the risk that the attack will be discovered. Less secure, is the network architecture itself, such as routers and switches which are often left unpatched due to a lack of redundancy and fears over availability [76, 25]. Consequently, attackers look to tap network links or install monitoring tools in poorly protected routers as a means to perform surveillance and reconnaissance [76, 25, 33, 4]. The capability to monitor all communications—also known as *pervasive monitoring*—is extremely powerful, to the extent that the Internet Engineering Task Force (IETF)[1] has declared that the mitigation of such monitoring should be an explicit goal of all IETF network protocols [20, 4].

For protocols that do not protect their payload, the entire content being communicated is available to the attacker. Thankfully, following the advice in RFC3365, most such protocols have a secure variant that encrypts the payload for confidentiality, and these are seeing ever-wider deployment [55, 4]. For example, SSL/TLS is a widely used protocol that enables two parties to establish a shared key to protect their transport layer payloads across an untrusted network. IPSec goes one step further and encrypts the transport layer headers as well. These protocols are a step in the right direction as an adversary can no longer trivially access the communicated content. However, the network layer headers (IP headers) are still exposed to the attacker. Knowledge of the source and destination address of a packet can reveal communication patterns, such as which users in the network are communicating and with what frequency. More worryingly, this knowledge can be used as the preliminary step to a more powerful set of attacks that can analyze encrypted traffic to extract sensitive information. For example, enterprise network managers will be worried to know that such attacks can be leveraged to listen in on VOIP conversations [71] or uncover user passwords from SSH sessions [73]. Similarly, this attack can also reveal the website being visited by the user of an ISP network [27, 19, 5, 37], the actions performed on

[1] The IETF is the body that governs network protocol standards such as TCP/IP.

an Android device [15] or the activities in a smart home [1].

Evidently, network layer privacy amongst untrusted infrastructure is of critical importance to all classes of networks. The Internet however, is a global network of independently managed networks and hosts. As such there is no central authority responsible for the operation or provision of security of the network [55]. Therefore, envisioning a silver bullet that is all encompassing and provides privacy across all networks is impractical. We believe that a more realistic vision for the future is for individual networks to bake in privacy as part of their offering. We agree with Raghavan et al. in claiming that it is in every network operator's best interest to provide *privacy-as-a-service* [53].

Unfortunately, the gold standard for network layer privacy, Tor [18], is not necessarily an ideal fit in this regard. Primarily, this is due to the usability issues associated with Tor, and overlay systems in general—high and variable latencies, which makes it unsuitable to the demands of latency critical applications such as VOIP, gaming, video streaming and so on. Most of this overhead can be attributed to an expensive circuit setup procedure as well as computationally heavy onion encryption operations performed on each message transfer. A knock on effect of the heavy computational demands imposed by Tor is that it requires well-resourced machines and cannot be implemented on a lot of commodity switches and routers [70].

We can model the problem of privacy within the domain of a single network as one of hiding protocol headers and payloads. As highlighted above, while Tor exists as a solution for network layer privacy, it comes with prohibitively high costs and is difficult to effectively deploy within a single network.

More recently, newer solutions have been proposed [28, 11, 12, 54] that tackle the problem of in-network privacy. Relative to Tor and other overlay solutions, these designs use light weight cryptographic operations and subsequently, offer lower latency and higher throughput. However, they still suffer from an expensive circuit setup phase. Moreover, these works propose clean-slate designs requiring next-generation Internet infrastructure and are not immediately deployable.

Rather alarmingly, despite the fact that both Tor and the other network-based solutions hide packet headers, they are susceptible to a number of the traffic analysis attacks described above [59, 50, 49, 66]. To understand why, we look to Juarez et al., who show that a critical requirement in these traffic analysis attacks is the ability for an adversary to isolate packets belonging to a target flow. In Tor and co., packet headers are pseudonymised. In other words, there is a one-to-one mapping between the plaintext and obfuscated packet headers. Therefore, while an adversary may not know the original header values, she is able to recognize and collect packets belonging to a single flow, thus enabling the traffic analysis attacks.

In light of these revelations, we revise our model for privacy within a network as one of not only hiding packet headers and payloads but also doing so in a non-injective (non one-to-one) fashion. In other words, any solution should look to break the one-to-one mapping between the plaintext and obfuscated packet headers.

We make use of this model to propose TrafFu—a new approach to hinder pervasive monitoring within a network using Software Defined Networks (SDNs). SDN takes the job of making routing decisions away from individual forwarding elements (FEs), such as routers and switches, and concentrates them in a centralized SDN controller for the network. In our system, TrafFu encrypts the source-destination IP addresses, and relies on the controller to decrypt them and make the appropriate routing decisions for the FEs. Thus, the original source and destination is never revealed to any FE and we succeed in our base requirement of hiding packet headers from compromised FEs. The key addition to TrafFu is that

unlike previous works, we look to increase the cost of traffic analysis attacks by reducing the number of packets that can be linked to the same flow. We achieve this by encrypting packets of a flow with different keys, thus providing for our other goal of non-injective obfuscation.

Previous works [62] claim that these types of defences are too difficult and expensive to be deployed in traditional networks, as routing traffic with an ever-evolving set of encrypted addresses will involve non-trivial network management challenges such as real-time global reconfiguration, and synchronization of several network devices in a decentralized environment, and as such, will prove disruptive and costly [29]. A big benefit to SDNs however, is that the controller acts as a central management unit for the network and can handle dynamic network conditions (key exchange, path diversity, topology changes, etc.) at a significantly lower cost and effort.

Another benefit to the controller being responsible for routing decisions is that unlike previous works, we can avoid a circuit setup phase altogether. Moreover, this also eliminates the requirement for FEs to perform any cryptographic operations. As a result, TrafFu is considerably more efficient and lightweight than its predecessors and can be immediately deployed on commodity network infrastructure.

1.1 Contributions

Concretely, our work makes the following contributions,

- We introduce TrafFu, a novel approach towards network layer privacy in an environment of untrusted FEs. TrafFu leverages SDNs to offer strong anonymity guarantees while allowing for flexible network management. Moreover TrafFu also improves upon previous offerings in terms of latency and bandwidth overhead while allowing for immediate deployability by working with commodity network infrastructure.

- We conduct a thorough security analysis and in particular, highlight the increase in difficulty for Traffic Analysis attacks.

- We further discuss the biggest cost associated with TrafFu—an increase in flow table utilization and demonstrate a solution to combine flow aggregation on encrypted IP addresses in order to reduce this cost.

- We prototype TrafFu and evaluate it against the increase in flow table utilization and the overhead involved. We show that the effect of flow table congestion can be offset by employing flow aggregation on encrypted addresses. Moreover, we show that the overhead associated with TrafFu is a significant reduction on other related works.

1.2 Thesis Structure

The remaining chapters are organized as follows. Chapter 2 provides a quick introduction to Software Defined Networks. Chapter 3 goes through some related designs in network privacy and looks at their corresponding attack and defence vectors. Chapter 4 sets the foundation for our discussion, introducing the attack model, assumptions and objectives. Chapter 5 is a comprehensive look at TrafFu, its design and discusses our solution to offset increased FE flow table utilization. Chapter 6 considers various attack vectors and evaluates TrafFu against them. Chapter 7 discusses our implementation, while Chapter 8

evaluates a tradeoff between FE flow table utilization and privacy before comparing TrafFu against related designs for overhead.

Chapter 2

Software Defined Networking (SDN)

Software Defined Networking (SDN) is a networking paradigm that aims at facilitating a programmable network architecture. Research in this area is vast and beyond the scope of this thesis. We limit our focus to principles, implementations and applications that are relevant to our topic of discussion. We begin by explaining the model for traditional networks and highlight their limitations before introducing the SDN architecture and how they look to overcome them. Finally, we discuss OpenFlow [40], the most popular SDN protocol.

2.1 Traditional Networks

The traditional network architecture of today can be represented as a distributed set interconnected switches and routers in which the control and data planes are co-located. The control plane refers to the network intelligence, i.e., calculating routes, managing access control etc. The data plane on the other hand refers to physically forwarding packets based on some specified policy. This model is illustrated in Figure 2.1.

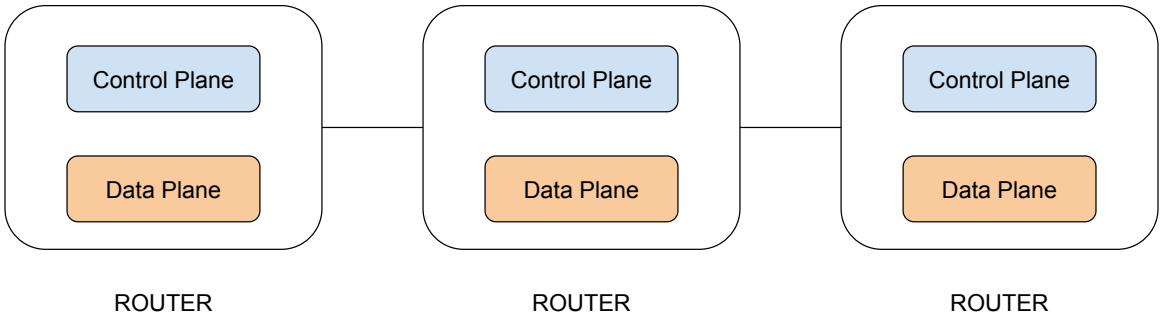


Figure 2.1: Traditional network model

Current trends in network usage are vastly different to before and significantly more large scale. This has exposed some basic limitations of the traditional network architecture. The system in place currently comprises of a myriad of different protocols designed to solve individual, specific problems [74, 60, 48]. These protocols are often unnecessarily complicated in order to effectively meet industry requirements

like availability, security and connectivity, amongst decentralized, heterogeneous infrastructure without any means of coordination [74]. This poses a big challenge for network management as even enforcing simple network policies would involve the configuration of hundreds of forwarding elements, which would not scale well to the complexity of today's networks, or require an overlay network to direct traffic through middleboxes purpose-built for a particular policy [48]. Simple tasks like adding or removing a router/switch would require the re-configuration of large portions of the network to account for access lists, routing policies and so on. The lack of any standardized open interfaces have led to static and inflexible networks that are unable to be customized to different environments [60, 48].

2.2 Software Defined Networking

As a response to these limitations, Software Defined Networking (SDN) was proposed as a new network architecture to facilitate a programmable control plane, by separating it from the data (forwarding) plane [35]. The control plane, i.e. network intelligence, is migrated into logically centrallized software controllers that *virtualizes* the network and abstracts the infrastructure for applications and services. The standard architecture of the SDN can be viewed in Figure 2.2.

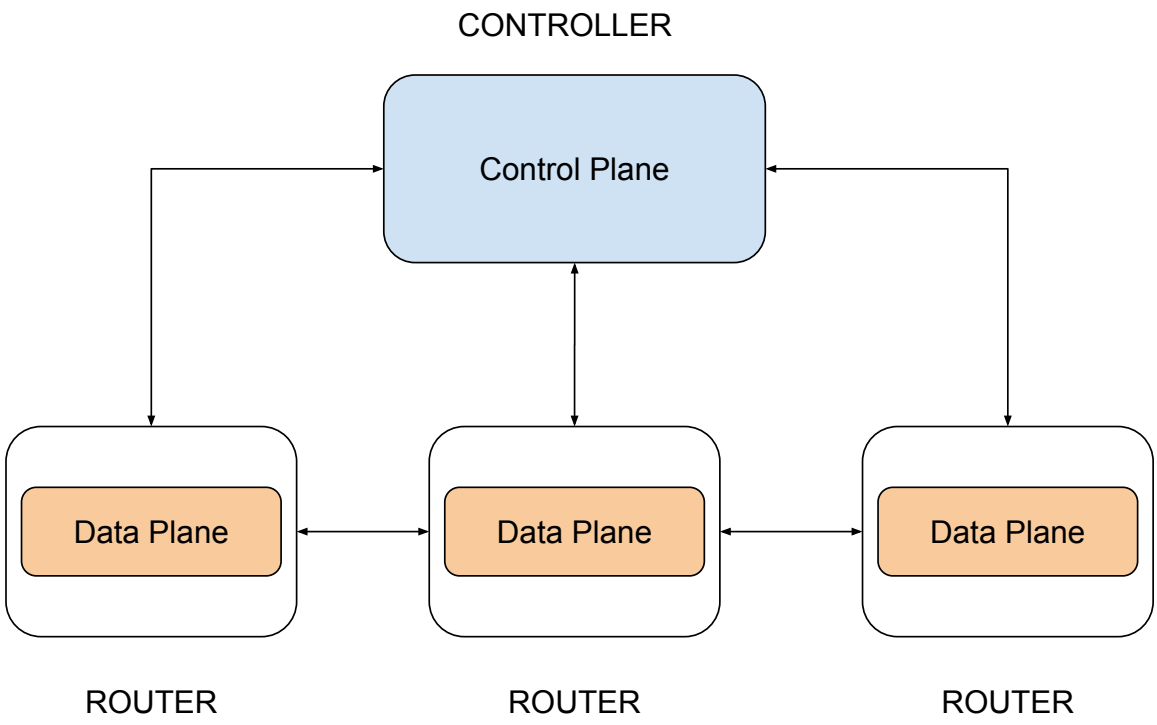
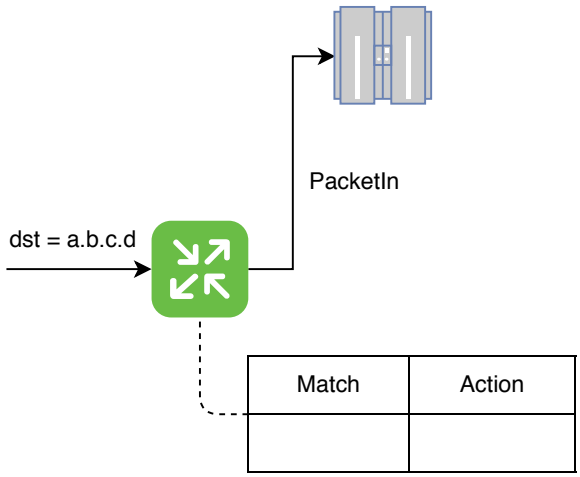
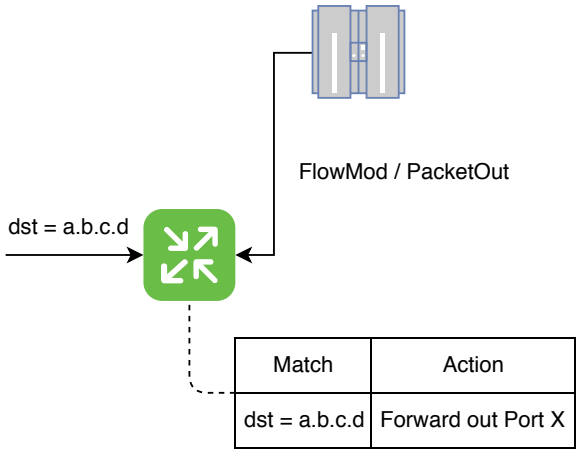


Figure 2.2: Software Defined Network model

This offers a multitude of benefits. The controller abstraction allows operators to view and manage the network as a single node and simplifies the process considerably. New policies, services and applications can be deployed at a much faster rate, while they can also be easily customized to suit the situation [60].



(a) OpenFlow switch forwarding a PacketIn request when incoming packet matches no flow table entry



(b) Controller sending a FlowMod or PacketOut message after processing the PacketIn message and installing a flow rule on the switch

Figure 2.3: Simple illustration of OpenFlow interactions between switch and controller

2.3 OpenFlow

The most popular and widely used implementation of SDNs is the OpenFlow protocol [40] which defines an open API between the data and control planes.

OpenFlow models forwarding state as a set of match-action rules, called *flow entries*. These rules comprise of a set of conditions over the packet header (IP address value, VLAN ID etc) and a list of actions which define the appropriate operations to be conducted on the packet (forward to port, rewrite MAC address etc). The flow entries are installed by the controller on a particular switch's *flow table*. A flow table is the fundamental entity of an OpenFlow switch. One or more flow tables house the flow entries used by the switch to make forwarding decisions on packets. If a packet matches all the conditions of a single flow table entry, the corresponding actions are performed. If a packet does not match any flow table entry, OpenFlow switches are configured to execute some *table-miss* actions such as dropping or forwarding the packet to the controller for further processing. In the latter case, the controller can then process the request and add the necessary flow entries to the relevant flow tables.

Figures 2.3a and 2.3b provide an example of how a forwarding element interacts with the controller in order to populate its flow table and forward a packet. We note here that forwarding elements in SDNs and in OpenFlow, are not tied to a particular implementation and can be implemented on legacy switches and routers, so long as they possess a channel to communicate with an external controller and one or more flow tables to lookup and perform actions on packets [40]. In Figure 2.3a, an OpenFlow compatible switch, receives a packet with a destination address of $a.b.c.d$. Since the flow table is empty, the packet does not match with any flow entry and the switch executes its table-miss action, which in this case, we take to be one that forwards such packets to the controller. Note that switches may have differently configured table-miss actions that drop or otherwise handle the packet. The OpenFlow *PacketIn* message is used by switches to send data packets to the controller. The controller analyzes and processes the PacketIn message to determine that packets at this switch with a destination address of $a.b.c.d$ need to be forwarded out of port X. The controller then uses the OpenFlow *FlowMod* message to update switch's flow tables accordingly. As seen in Figure 2.3b, the switch now has a flow entry that matches with every packet destined to $a.b.c.d$ and forwards them out of port X. The controller can also use a *PacketOut* message to inject the packet into the dataplane of the switch and forward it out of a particular port.

Chapter 3

Related Work

The surveillance of network traffic is a topic that is at the forefront of public consciousness today. Naturally, it has garnered significant interest from researchers and there has been a tremendous amount of work done across its entire spectrum. While it would be nigh impossible to cover this body of work in its entirety, this chapter looks to provide the reader with a big picture overview of various threats and mitigations as applied within the context of our project. We begin by surveying the state of unprotected traffic and a few proposals designed to provide transport layer privacy. In highlighting its deficiencies, we set the scene for our requirement of network layer privacy. We continue by describing existing solutions and proposals for network layer anonymity. Following this we introduce the concept of Traffic Analysis attacks. We highlight the depth of its power by showcasing the many advances made in only one type of this attack. We then try and extract a high level understanding of the key requirements to successfully conduct a traffic attack and show how this understanding forms the basis for our defence. We conclude this section with a few proposals that also leverage SDNs to preserve network privacy.

3.1 Protecting Web Traffic

From storage, to gaming, to computational workloads, we are seeing an ever increasing set of activities being ported to the Internet. For all the benefits this provides, the ubiquity also ensures that network traffic today is a goldmine for malicious parties interested in gathering more information about web users. As highlighted in Figure 3.1, unprotected traffic gives an adversary eavesdropping on network infrastructure (links, routers, switches etc) easy access to its information.

In response, a number of proposals have emerged that look to protect the contents of packets being transmitted across a network [22, 17, 3]. One of the most popular today is the TLS protocol [17]. The high level overview of the protocol is presented in Figure 3.2. A client wanting to setup a TLS connection signals its intent to the other party. The two parties then authenticate each other before establishing a shared key. The process of establishing a shared key is termed as the *TLS Handshake*. This key is subsequently used to protect the contents of packets being transmitted across the network between the two parties.

However, while the protocol does protect the payload, higher level layers, such as the network (or routing) layer is left unprotected. Thus, following on from the previous example while an adversary may not be able to know User A bought from Amazon, it can still know that User A was communicating

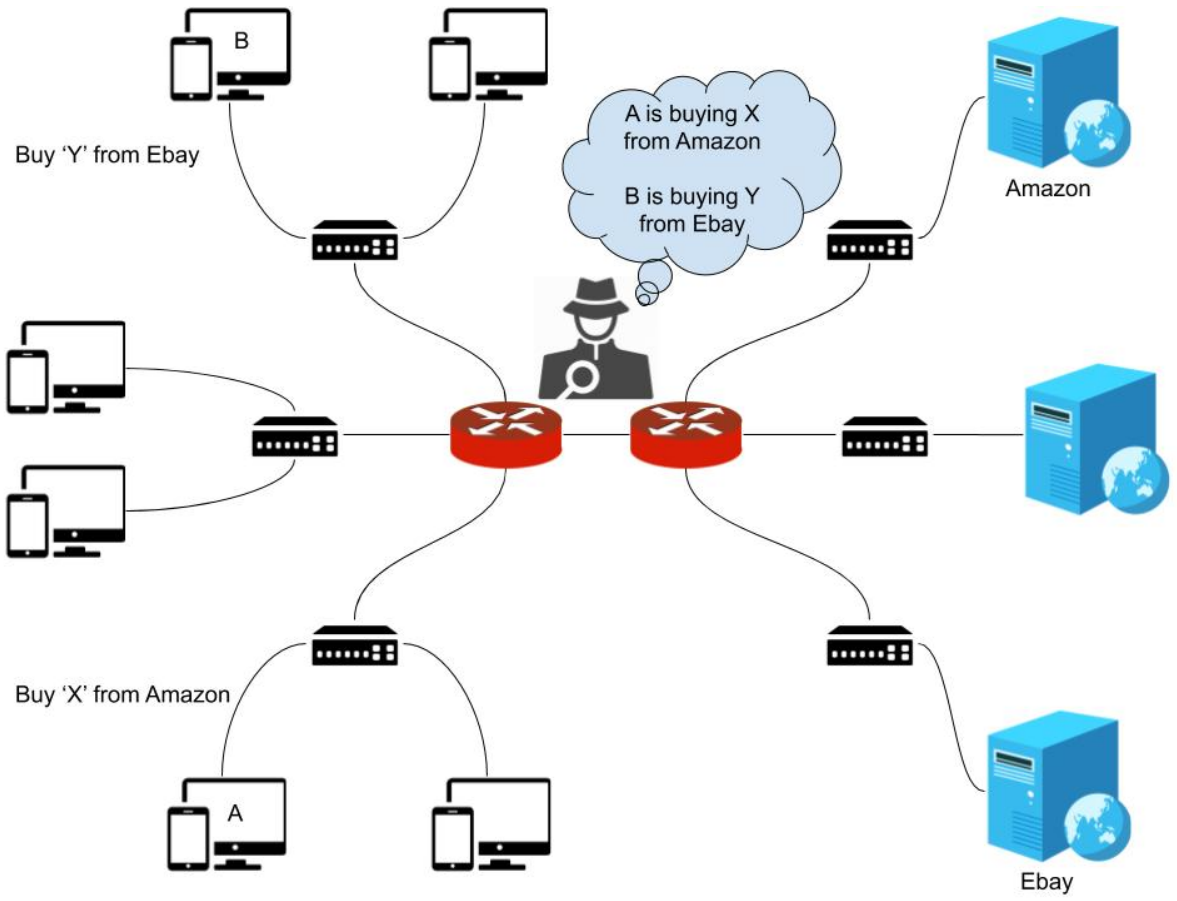


Figure 3.1: Unprotected traffic gives eavesdroppers easy access to information

with Amazon. This is a valuable source of information to an adversary. Knowledge of the source and destination address of a packet can reveal communication patterns, such as which users in the network are communicating and with what frequency. More worryingly, this knowledge can be used as the preliminary step to a more powerful set of attacks that can analyze encrypted traffic to infer sensitive information. We discuss these attacks in greater detail in Section 3.3.

3.2 Routing layer anonymity

The security community took note of the problems caused by not protecting network layer headers. Subsequently, many designs have emerged that look to address this concern.

3.2.1 Proxies and Mix-Nets

One of the seminal works in this domain was Chaum's original Mix-Net [10], to which all modern proxy/mix-net designs can be traced back to. This work proposes passing a message through a series, or cascade, of *Mixes* with the message wrapped in multiple layers of encryption. Each mix subsequently decrypts once (removes one layer) and forwards packets based on some mix strategy. Designs such as Mixmaster [45] and Mixminion [16] are similar, but differ primarily by employing different mix strategies

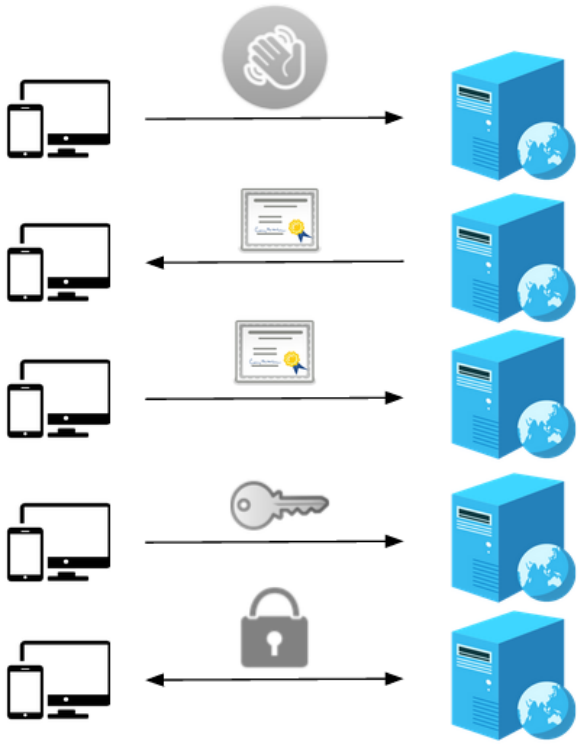


Figure 3.2: Overview of the setting up a TLS connection

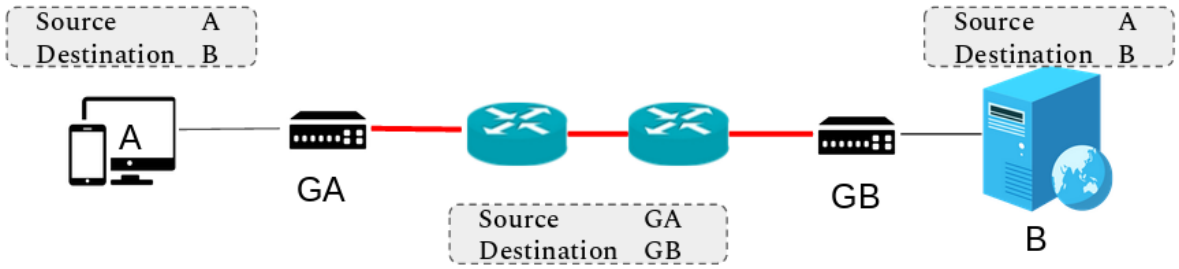
that provide stronger anonymity guarantees. Most Mix-based systems are designed as overlays and provide strong anonymity guarantees at the cost of high latency. Therefore, they are limited in their use case to non-interactive applications such as email, wherein latency constraints are relatively looser.

Another popular design that has emerged in the recent past is the IPSec protocol [3]. IPSec comes in many flavours, so we restrict our focus to the most secure implementation. IPSec in tunnel mode routes packets through a channel setup between two gateway servers. As can be seen in Figure 3.3a, while in this tunnel, the regular routing headers are replaced with addresses of the gateway servers (GA and GB). Further, as highlighted in Figure 3.3b, the packet format for the encapsulated security protocol in IPSec allows for confidentiality and authenticity of the packet received at GB.

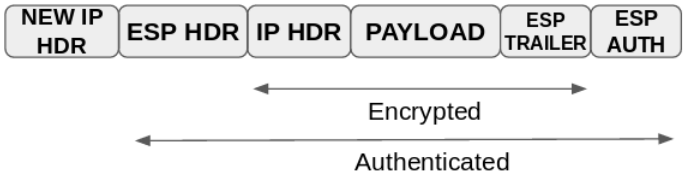
IPSec is a considerable improvement on Mix-nets in terms of latency. However, they share the same weakness of a rigid routing scheme that mandates flows to pass through the same (or same set) or gateway servers. Moreover, network level headers are exposed before and after the gateway servers and can be exploited by attackers in a similar fashion to TLS packets.

3.2.2 Non-proxy based solutions

A number of solutions took an alternative approach to proxy networks. Onion Routing [63] was designed to provide relatively strong anonymity guarantees at lower latencies than Mix-nets. Tor [18], the gold-standard in this domain, operates similarly to Mix-Nets in terms of multiple encapsulated encryptions, but differs by not using the same sequence of Mixes. Instead, the Tor network, which comprises of a multitude of relays called *onion routers*, assigns some number of these routers (often 3) to a flow, to form a bidirectional circuit. To improve latency, Tor employs no delay/mix strategy for incoming packets,



(a) Packet Header transformation across the IPSec tunnel



(b) Encapsulated packet format in IPSec

Figure 3.3: IPSec tunneling protocol

however, the cost of setting up a circuit for each flow, the multiple encapsulated encryptions on each message and the inherent costs of an overlay system means that Tor still exhibits a non-trivial amount of overhead and is not suited for interactive applications.

Recently, a number of works have abandoned the overlay approach and have instead proposed purely network-based anonymity solutions [12, 28, 11, 54]. These were designed for high throughput and low latency at the cost of comparatively weaker anonymity guarantees. Fundamentally, these designs require a) intermediate routers to be able to perform at least lightweight symmetric cryptographic operations and b) forwarding state to be embedded in the packet rather than the network device.

Hornet [11] uses an onion encrypted scheme similar to Tor and therefore incurs circuit setup and on-path node verification latency. Moreover, to prevent storing any session state, each intermediate node stores its encryption keys in the packet header. This increases packet size and subsequently, reduces system throughput. Moreover, Hornet requires source routing, i.e., the source is required to select paths. Thus, the system increases the computational load on the routers by requiring onion encryption operations, while also increasing the load on the client by requiring it to be topologically aware and compute routes for flows.

LAP [28] is more lightweight in this regard and avoids onion encryption and source routing. In this design, during circuit setup, each intermediate Autonomous Domain (AD) constructs a segment containing its routing decision (egress interface) for the given destination. The AD then encrypts this segment using a private symmetric key and appends it to the packet header. This segment is then extracted and decrypted when next receiving packets of the same connection. However, since the ADs are responsible for route computation, the destination must be exposed to at least some fraction of the nodes in the network during circuit setup. LAP keeps it simple and does not hide the destination from any node during this phase. This has implications on privacy as a malicious first hop AD can trivially deanonymize the source and destination, thus breaking any anonymity guarantees.

Dovetail [54] addresses this issue by using a matchmaker node to hide the destination from the first hop AD. In this design, the source picks a matchmaker node and constructs a path to it. In the process, it also passes the identity of the destination to the matchmaker. Thus, the nodes until the matchmaker node only know the identity of the matchmaker node, while the matchmaker node only knows the destination and not the source. The matchmaker node then proceeds to construct a path to the destination in a manner similar to LAP. Apart from source routing, Dovetail has another drawback (which it shares with LAP) in that it leaks path information such as the path length and the position of the current node on the path, which significantly degrades the anonymity set for topology-aware attackers. This is due to the fact that path segments are appended to the packet header in a deterministic fashion. Therefore, a malicious node can look at the position of its own segment relative to others, and make some inference about its position on the path of the flow.

PHI [12], randomizes path segment positions to prevent leaking path information. It also relieves the source from the burden of path selection without compromising on any anonymity guarantees relative to LAP and Dovetail. However, this requires additional midway and helper nodes to be part of the circuit setup, resulting in relatively higher latency and inefficient routing policies. Moreover, randomizing path segments can often result in failure during path setup and needs it to be retried. This further exacerbates the latency overhead involved.

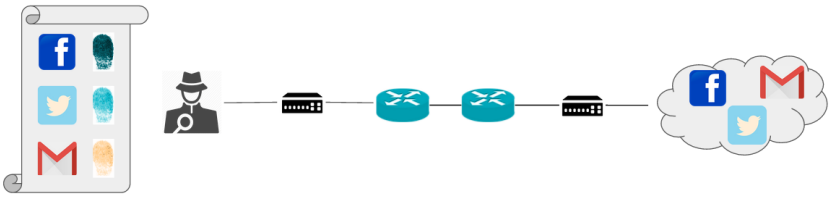
3.3 De-anonymization using Traffic Analysis

The previous section introduces some existing solutions for network layer privacy. Performance issues notwithstanding, these designs highlight the potential for obfuscating netork layer headers from malicious nodes within the network. However, this obfuscation only prevents eavesdropping from direct observation of packet headers. *Traffic Analysis* is an inferential method to extract information embedded in network traces by analyzing the properties observed in the traffic [14]. The scope of such attacks are vast and can range from extracting the type and frequency of websites being visited, to exposing the actions performed on a smartphone to uncovering operating system of the device involved. Note that these attacks can be conducted on packets with obfuscated network level headers as well. Thus, all of the previous designs mentioned are susceptible to this attack. For the purposes of this thesis, it suffices to get a feel for the potential for such attacks. Consequently, in the following section, we pick one such attack, *website fingerprinting* and explore its body of work in detail.

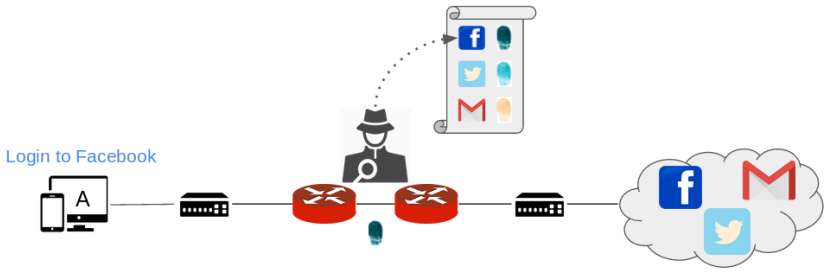
3.3.1 Website Fingerprinting

Website fingerprinting is an attack where a passive adversary, eavesdrops on encrypted traffic at some point between the source and destination, and based on patterns observed in the traffic, makes inferences on one, or both ends of the connection. This is a powerful tool for mass surveillance as knowledge of the communication endpoints can reveal, among other things, information about the type and frequency of websites being accessed, services being used and information being requested.

A very simplified breakdown of how website fingerprinting works can be observed in Figure 3.4. As a preliminary step, an adversary looking to conduct this attack establishes flows with some set of target websites. It then observes the pattern, or *fingerprint*, of traffic generated and stores it in a database (Figure 3.4a). In the next step, the attacker eavesdrops on traffic at some point in the network and based on the observed traffic fingerprint, makes a conclusion on the website being visited (Figure 3.4b).



(a) Preliminary step for website fingerprinting



(b) Inferential step for website fingerprinting

Figure 3.4: High level overview of website fingerprinting attacks

This section first goes into a depth first dive into the various website fingerprinting attacks conducted over anonymous networks like Tor, before looking at defences that have been proposed and their relative degrees of success.

Fingerprinting on Encrypted Connections

The first recorded analysis of encrypted traffic was by Cheng and Avnur [13] as they attempted to infer the file being accessed over SSL from a known server. The pattern or fingerprint used in this case was file size information. Hintz and Sun both enhanced this work by targeting websites when the server was not known (HTTP proxy servers). Sun [61] proposed the Jaccard's coefficient as the basis for his classifier and was able to identify 75% of the sites visited in a sample space of 100,000 websites, with only a 1.5% False Positive Rater (FPR). Both these attacks relied on the presence of an up-to-date database of object-number and object-length profiles of the target pages. However, the advent of HTTP pipelining, persistent connections and tunnel-based Privacy Enhancing Technologies (PETs) such as VPNs, OpenSSH tunnels, etc made it hard for outsiders to infer file sizes thus rendering these attacks all but redundant today.

Bissias et al. [5] moved away from this approach and instead, focused on IP headers and inter-packet arrival times, thus, allowing it to be generalized towards even tunnel-based PETs. Liberatore and Levine [37] on the other hand, chose to compare packet size histograms with the Jaccard coefficient and a Naive Bayes classifier with kernel density estimation. They achieved a classification accuracy of 73% over 1000 websites, but were significantly worse off in the face of packet length padding. Lu et al. [39] highlighted that considering features like packet ordering as part of the fingerprint can improve the classification accuracy.

Fingerprinting on Anonymous Networks

Herrmann et al. [27] were the first to apply packet-based fingerprinting on popular PETs. Using a multinomial naive Bayes classifier, they relied on a fingerprint that included packet features such as size frequency distribution and direction while neglecting others such as order and timing. With a sample space of 775 index pages, they displayed improvements over previous classifiers such as Jaccard's and Naive Bayes, obtaining accuracies in excess of 90% for single hop systems. For multi-hop systems like JAP and Tor, their accuracy was much lower at 19.97% and 2.96% respectively. This degradation is largely down to the ability of systems like Tor to send data in fixed size cells, which renders the packet size frequency distribution feature redundant.

Panchenko et al. [50] made significant improvements to this using Support Vector Machines (SVM). Using the same dataset as Herrman et al., this approach was able to recognize approximately 55% of webpages accessed over Tor. Additionally, the authors were the first to evaluate the efficacy of fingerprinting attacks in an open-world scenario. Previous works operated on a closed-word scenario which unrealistically limited the websites a user could visit to a fixed number. In the more-realistic open-world scenario, the user is allowed to visit as many websites as possible and the adversary only tries to determine if the website belongs to a some fixed set of target/monitored websites. They achieved a 73% accuracy in this scenario.

Cai et al. [7] also used an SVM but instead of manually specifying features, implemented an edit distance algoritm to automatically extract them from communication traces. They achieved a recognition rate of 70% over 800 webpages. Additionally, they were the first to identify full websites rather than webpages, by proposing a Hidden Markov Model extension to this attack. Wang et al. [67] tried to improve upon this by statistically eliminating Tor flow control cells, improving the quality of the training set and achieved in excess of 90% for both closed- (100 webpages) and open-world (1000 webpages) scenarios. However, both [67, 7] impose high computational cost on the attackers while [67] has been further criticized for its small open-world size and operating on a broken implementation of Tor's Browser [51].

Since then, attacks have been developed that not only keep the classification accuracy high, but also lower the cost involved. Wang et al. [66] used a novel k-Nearest Neighbors (kNN) classifier that uses a distance metric to measure similarity between different websites. This reduced the time required for training and achieved good results in both closed-world (91%, 100 pages) and open-world (86% true positive rate (TPR), 0.6% false positive rate (FPR), 5000 pages). Panchenko et al. [49] proposed CUMUL, a novel attack that abstracted the loading process of a webpage by generating a cumulative behavioral representation of its trace from which features were extracted. Each feature instance was represented as a 104-coordinate vector formed by the number of bytes and packets in each direction and 100 interpolation points of the cumulative sum of packet lengths, accounting for direction. In a closed setting, the attack achieved 91% accuracy. In the open-world, they evaluated the attack on two scenarios - multi-class, where every monitored page was in its own class and two-class, where the whole set of monitored pages is in a single class. The TPRs for both classes were 96% whereas the FPRs were 9.61% and 1.9% for the multi- and two-class cases respectively. This work also is one among a few that take background traffic into account. Hayes and Danezis [26] proposed the k-fingerprinting attack (k-FP) that used a random forest classifier trained on regular features to extract fingerprints. k-FP achieved an accuracy rating of 91% in the closed setting and had an 88% TPR, 0.5% FPR in the open setting. This work additionally conducted a study that used a set of 175 features, including variations

of those commonly used in literature such as size, timing and so on, to discover the most incriminating features and ranked them. They concluded that for most attacks and classifiers, simple features like packet counting revealed more than complex ones such as interpacket arrival time or packet ordering.

The current state of the art in website fingerprinting comes from the work of Sirinam et al. [59] that introduces DeepFingerprinting (DF), an attack that leverages CNN-based deep learning techniques. A powerful and sophisticated model means that the attack requires simple input and does not need to handcraft features for classification as it can do that automatically. It achieves a 98.3% accuracy in a closed world setting of 95 pages. In an open-world setting of 20000 unmonitored sites, DF achieves 0.99 precision and 0.94 recall. They also examined the efficacy of their attack on traffic using state of the art fingerprinting defenses namely WTF-PAD [32] and W-T [69] (introduced next section).

Fingerprinting Defences

Most of the earlier work in this domain considered packet lengths to be the most incriminating side channel to base traffic analysis upon. Subsequently, different padding schemes were explored as potential countermeasures for fingerprinting. Liberatore and Levine [37] simulated a number of such techniques and found that it significantly degraded the accuracy of their attacks. Wright et al. [64] proposed Traffic Morphing, which looked to modify a trace to statistically resemble the packet length distribution of a different webpage.

Dyer et al. [19] proved that these countermeasures were ineffective against fingerprinting attacks such as [27, 50]. Further, they concluded that hiding packet lengths is pointless as an attacker only needs coarse grained information such as total time, bidirectional bandwidth and traffic bursts to distinguish websites. Moreover they conceptualized BuFLO, a high bandwidth approach to reduce the information available to an adversary, by sending fixed size packets at fixed intervals.

Cai developed CS-BuFLO [7] and Tamaraw [6] as alternatives to BuFLO. CS-BuFLO was more practical in design and reduced bandwidth overhead while also providing rate adaptation and congestion sensitivity. Tamaraw grouped pages into anonymity sets and generated padding based on webpage size. It also treated incoming and outgoing packets differently. Glove [46] is an SSH based approach that clustered pages into large similarity groups and added some amount of cover traffic such that sites within a group were indisinguishable. A drawback of these approaches is that they came with high levels of overhead. Sirinam et al. [59] report that on average, the BuFLO based approaches came with 130% bandwidth overhead and loaded pages two to four times slower than vanilla Tor.

The current state of the art fingerprinting defences are WTF-PAD [32] and Walkie-Talkie [69]. WTF-PAD uses Adaptive Padding, a method first introduced by Shmatikov and Wang [58], to vary padding based on network usage as a means to hide coarse grained features. WTF-PAD incurs significantly less overhead than BuFLO based defences and was previously considered to be secure against all attacks until DeepFingerprinting [59] experienced success in fingerprinting Tor traffic employing WTF-PAD as a defence. Walkie-Talkie (W-T) on the other hand, operates in half-duplex mode, i.e., the client and server only send non-overlapping bursts of traffic in each direction. Cover traffic and delays are inserted to create collisions of the trace features with other websites. The key idea here is that creating collisions on non-overlapping traffic requires less padding and as such, is more efficient. DeepFingerprinting was also mildly successful against W-T but to a lesser degree than WTF-PAD.

3.3.2 Key Requirement for Traffic Analysis

The attacks we described above look to extract information based on patterns observed in the traffic for a flow. Similarly, the defences proposed look to hide the traffic patterns of a flow. The key insight here, as highlighted previously [72, 31, 68], is that this requires the adversary to know what packets belong to a target flow. Only when an adversary can isolate the packets belonging to a flow, can she look to extract/observe a fingerprint. With the exception of high latency Mix designs, the anonymity solutions described above do not prevent (or hinder) the adversary from isolating her target packets. This is because in these designs, packet headers are pseudonymised. In other words, there is a one-to-one mapping between the plaintext and obfuscated packet headers. Therefore, while an adversary may not know the original header values, she is able to recognize and collect packets belonging to a single flow, thus enabling the traffic analysis attacks.

Tor headers contain a *circuit id* used by onion routers to look-up the next hop along the path. While this doesn't reveal the original identity of the source or destination, it can be used to identify packets belonging to the same flow. The Anonymous Header used in Hornet packets can be leveraged similarly. The enrypted header segments in LAP and Dovetail are also such indicators. In PHI, the headers contain a public key hash that act as identifiers unique to the session they represent.

Subsequently, these designs are inherently vulnerable to powerful traffic analysis attacks. As will be shown throughout the length of the document, we look to hinder this fundamental ability of an attacker and eliminate session identifiers from packet headers — making it hard to collect and isolate packets of a single flow.

3.4 Anonymity using SDNs

The advent of SDN and Openflow has garnered much interest from the research community, including in their applicability for network privacy. Mendonca et al. proposed AnonyFlow [42], a user-transparent, in-network anonymization service. In this design, each host communicates with its destination server using a pseudonymous ID and the central controller stores a mapping between the real and pseudonym IDs. The paper pits the end server as the adversary and thus, does not hide the destination from the forwarding elements. Moreover, given that each source is associated with a particular pseudonym that is typically not modified within the duration of a flow, it makes isolating packets belonging to a flow relatively easy and ensures that Anonyflow is vulnerable to traffic analysis attacks.

Jafarian et al. [29] try to defend against external and internal scanning attacks with a moving target architecture that randomly mutates end-hosts' IP addresses to unused virtual addresses. The SDN controller is used to maintain the mapping between real and virtual IP addresses. The design is fairly resilient from a basic anonymity perspective — no forwarding element finds out both the original source and destination address. However, like [42], the mapping between real-virtual addresses are static and unchanged within a flow and thus, can be leveraged to execute traffic analysis attacks.

SPD-IMAM [77] proposes a very similar scheme, relying on a mapping of real-virtual addresses, but rewrites the source-destination addresses at each hop, to new virtual addresses. Like Anonyflow, the first-hop node observes the destination in plaintext and can trivially deanonymize flows. A non-first hop node, the address rewriting notwithstanding, does not see the original addresses, but always sees a static virtual address pair thus allowing for website fingerprinting attacks.

Chapter 4

Attack Model and Objectives

The following section lays out the parameters that we base our design on. We begin by describing our model for the attacker in terms of their motivation and capabilities. We then introduce the other party—the defender seeking to thwart our adversaries efforts. Simultaneously, we also state any assumptions we make in our definitions. This discussion gives us the ability to outline certain objectives, which we present at the end.

4.1 Attack Model

Similarly to related proposals [28, 12, 11, 54], we model our attacker as an idealized, external adversary seeking to conduct passive surveillance against a network. In keeping with recent trends [76, 25, 33], we assume the adversary does not focus on targeting internal system machines, but rather, looks to tap network links or install monitoring tools in poorly protected routers as a means to perform passive surveillance and reconnaissance. Similar to what is outlined in [4], the attacker has the ability to observe every packet (in-transit or at rest) of every flow passing through a node compromised by the attacker. However, given our model of passive surveillance, the attacker can take no action with respect to these communications (eg: block, drop, inject etc).

Given that the focus of our project is on network layer privacy, we restrict the scope of our attacker's interest to passive surveillance of network layer packet headers. We consider any information that can be obtained by direct observation of other layers, such as the transport layer, application layer or even packet payloads, as beyond the scope of this project. We assume that best practices are employed when protecting these layers and an adversary is incapable of decrypting payloads or headers that are already encrypted. We note that information may be leaked during the bootstrapping of such protection (i.e. certificate exchange during TLS). We discuss this in Chapter 5.6.

As outlined in Chapter 1, network level headers can not only reveal communication patterns, such as which users in the network are communicating and with what frequency, but can also be used to extract sensitive information from encrypted traffic. Ultimately, we realize that the **goal** for our adversary is to isolate packets belonging to a single flow [31].

If network level headers are unprotected (eg: TLS/SSL, IPSec), it is trivial for an attacker to do so. If network level headers are obfuscated (eg: Tor, LAP, Dovetail etc), but in an injective manner (i.e., obfuscated source-destination addresses map to the same pseudonymous addresses), then the adversary

can infer that all packets with the same pseudonymous headers belong to the same flow. However, if two packets from the same flow map to different pseudonymous headers, then the attacker, without better probability than pure guessing, cannot use the network headers to deduce that these packets belong to the same flow.

4.2 Defence Model

As outlined in Chapter 1, we model our defender as the operator of a single network domain wary of the threat posed by the external attacker. The operator is looking to offer network layer privacy without having to trust the network switches and routers. We acknowledge that this is a slightly unconventional outlook as the security community has traditionally viewed network operators with suspicion. However, we believe that there is plenty of incentive for the network operator to bake in network layer privacy as part of its offering. Cloud network providers would certainly like to protect their client's SSH passwords [73] while ISPs and wireless networks would surely want to prevent the leakage of actions being performed on a smartphone or a smart home [15, 1]. As such, we believe that it would be in an operator's best interest to provide *privacy-as-a-service* [53].

4.3 Assumptions

To simplify our analysis of TrafFu, we make the following assumptions

- As mentioned earlier, we restrict the focus of our adversary to network layer inferences. We assume that the higher level protocols are suitably protected using current best-practices and consider any information leakage from such layers as beyond the scope of this paper.

- We make a further assumption regarding the scope of our protection. We claimed earlier in Chapter 1 that the Internet is a global network of independently managed networks and hosts with no central authority managing network security [55]. This scenario does not lend itself well for a silver bullet solution that provides end-end network layer security across different networks. Consequently, we restrict the scope of TrafFu's protection to the domain of a single network as managed by a single operator.

- As will be explained in Chapter 5, we entrust the network management entity, and consequently the SDN controller, with maintaining the correspondence between plaintext and encrypted source-destinations. If this entity is compromised, all privacy guarantees in our system are broken, and thus must be incorporated into our Trusted Computing Base (TCB). As a logically isolated entity, the controller is akin to an internal system machine, which as we have established, are more frequently hardened, patched and monitored. Thus, it would be difficult to compromise the controller and an attacker would be best served targeting the FEs.

- We reiterate here that the primary goal of an adversary is to conduct mass surveillance and not targeted surveillance on select source-destination pairs. In the latter, an attacker can simply compromise the two terminal routers on the path between the target pair and would be very difficult to thwart. In the former however, the primary objective is not to target a single (or small set) of target pair(s) but rather, to conduct large scale data trawls to uncover as much information

about the network traffic as possible. This lends itself to the belief that an adversary will typically look to insert itself into a number of high-capacity links or a set of routers placed at strategic locations that provide access to a large sampling of network traffic [4, 56].

4.4 Objectives

Now that the parameters of our design have been established, we can outline the objectives for TrafFu

- **Sender and Receiver anonymity (O1)**: Pfitzmann and Kohntop determined anonymity as being unidentifiable within a set of objects, called the anonymity set [52]. Sender anonymity is defined as being unable to tell with better probability than pure guessing, who the sender of the message is. Similarly, receiver anonymity is the analogue for the message recipient. Given that we do not employ any cover traffic or constant rate padding schemes at the moment, we aim for sender anonymity on all FEs except the first hop. Similarly, we also aim for receiver anonymity on all FEs barring the last hop. This gives us the base level of protection for network level headers and prevents them from being exposed directly to compromised FEs.

- **Increase the cost of Traffic Analysis (O2)**: TrafFu should build upon **O1** to ensure that any obfuscation is applied in a non-injective fashion in order to hinder the ability of an attacker to isolate packets belonging to a flow.

- **Low bandwidth and latency overhead (O3)**: TrafFu should prove a significant reduction on previous designs in terms of overhead associated with latency, bandwidth and cryptographic operations.

- **Commodity Infrastructure (O4)**: TrafFu should rely on commodity SDN infrastructure, rather than custom routers such as those used in Tor.

Chapter 5

Design

We describe several different iterations of TrafFu, each building upon the other and attaining more of the objectives listed in Chapter 4.4. We begin by introducing a preliminary solution that is conceptually simple, requires no changes to the current SDN model and meets objectives **O3** & **O4**. However, it requires trust in the *edge* FEs and is susceptible to traffic analysis[1]. We then propose improvements that eliminate (or hinder) both of those shortcomings, thus fully attaining **O1** and **O2**.

5.1 Preliminary Design

5.1.1 Network Model

Our preliminary solution assumes a standard setting for an SDN with FEs connected in a routing network. One interface on each FE is reserved for its connection to the network management entity, the SDN controller. We assume that this connection is authenticated and secured using protocols like TLS and isolated from transit traffic. The edge routers, as introduced earlier, also use some interfaces to connect to one or more hosts. We further assume that the controller has a northbound interface that supports applications, where we deploy the TrafFu SDN application, *CryptoApp*.

5.1.2 Packet Forwarding Process

We illustrate how our preliminary design tries to provide anonymity for its clients in Figure 5.1. In it, we show the process by which a client $H1$ (IP Address $h1\_ip$) communicates with $H9$ (IP Address $h9\_ip$). We assume that these clients have not communicated before. In keeping with the figure, we outline the steps as follows

Step 1: Any application on $H1$ seeking anonymity forwards its packet with destination address $h9\_ip$ to its first hop switch, $S1$.

Step 2: Given that $S1$ does not currently have a flow rule or forwarding policy for packets destined to $h9\_ip$, it forwards a *PacketIn* request to the controller, asking for instructions on how to handle this packet.

Step 3: The controller uses the PacketIn message to determine that $S1$ received a message from the interface connected to $H1$ (it could also look at the packet header) and forwards it to the CryptoApp,

[1] Edge FEs in this case refers to the terminal (first and last hop) FEs along the path of a flow.

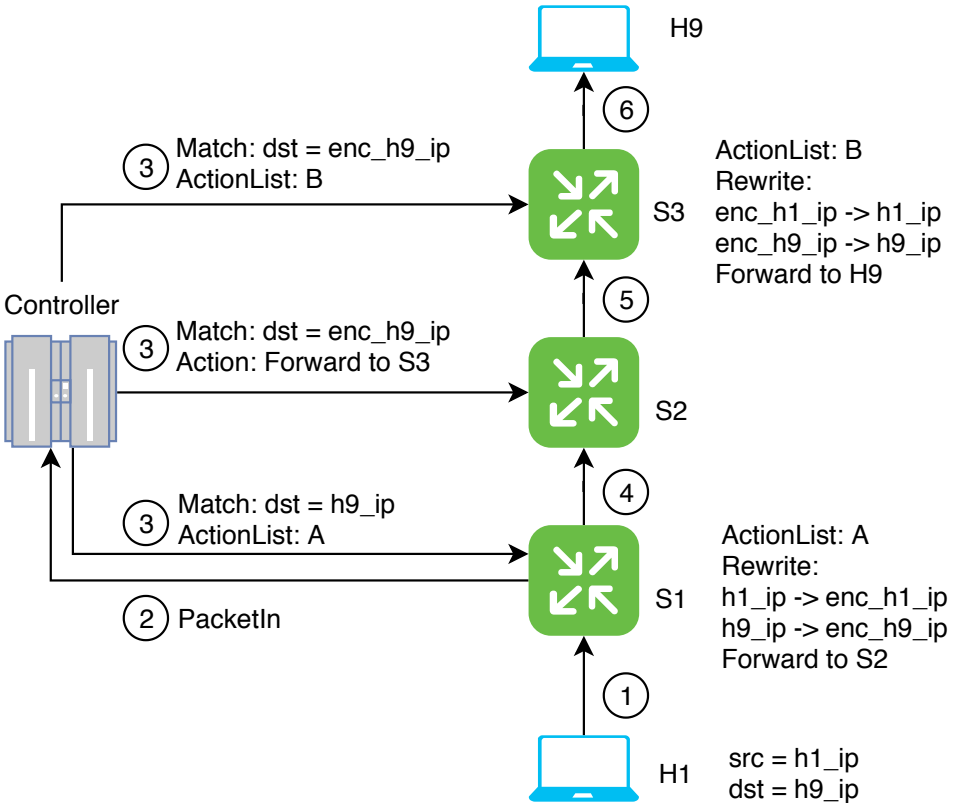


Figure 5.1: Client communication in preliminary solution

which encrypts and returns the encrypted source-destinations, enc_h1_ip and enc_h9_ip. The controller then computes a path from $H1$ to $H9$ and sends out flow instructions to each switch on the path, $S1$, $S2$ and $S3$. We employ the edge routers ($S1$ and $S3$) to act as *Rewriter Nodes*. On every packet of this flow, $S1$ is instructed to rewrite $h1\_ip$ and $h9\_ip$ to enc_h1_ip and enc_h9_ip respectively. Similarly, $S9$ is instructed to rewrite these addresses back to their original values ($h1\_ip$ and $h9\_ip$). $S2$ is instructed to simply forward packets destined to enc_h9_ip accordingly. The controller then instructs $S1$ to retry handling the packet.

Steps 4,5,6: After receiving its instructions from the controller, each switch blindly does as instructed. On every packet for this flow, $S1$ and $S3$ rewrite the addresses, while $S2$ simply forwards it to the next hop node.

$H9$ receives the unencrypted packet, unbeknownst to the in-network transformations. The next packet in this flow will hit the match field on $S1$ and trigger the corresponding Action List A, without controller intervention.

5.1.3 Discussion

The biggest benefit of this design is that it is transparent to end-clients and compatible with existing SDN architecture. This makes it readily deployable and helps us achieve **O4**. In addition, the scheme adds minimal overhead as there is only one encryption operation per flow (the very first packet). Moreover, we do not require a dedicated circuit setup phase. Thus, we can claim to satisfy **O3** as well. An added benefit is that for hosts that do not wish to have privacy protection, the SDN controller simply does not

invoke the CryptoApp and routes the packets as normal.

However, the design also has a significant drawback. The network layer addresses exposed to the edge (rewriter) nodes. This means that a compromised edge FE can simply observe packet headers and achieve its goal of isolating all packets belonging to a flow. Thus, we fail to achieve **O1**.

Additionally, this design also does not protect against an adversary located at an intermediate node ($S2$). The address encryption scheme is injective, i.e., a one-to-one plaintext to ciphertext correspondence. As such, while such an adversary would be unable to trivially uncover the original source and destination addresses, it can once again, easily achieve its goal of isolating all packets belonging to a flow. Thus even in the absence of a compromised edge FE, we fail to achieve **O2**. Subsequently, we propose modifications to our preliminary solution that improves its guarantees toward **O1**.

5.2 Design for improved anonymity

5.2.1 Network Model

In order to eliminate the need to trust edge FEs, we extend the network model described previously by a) adding an external facing interface on the CryptoApp that communicates with the outside world and b) requiring the end-hosts to encrypt the packet before sending it into the network.

When a client joins the network, it sets up a secure channel with the CryptoApp before establishing a shared key. The CryptoApp stores this key in a lookup table, indexed by the hostname. Note that the channel from the client to the CryptoApp can be via the the routing network itself and does not require a separate network.

5.2.2 Packet Forwarding Process

Once again, we explain this design by way of an example as shown in Figure 5.2. We consider the same flow and make the same assumptions as previously.

Step 1: $H1$ begins by encrypting the packet's source and destination address to $enc\_h1\_ip_{H1}$ and $enc\_h9\_ip_{H1}$ respectively before sending it to $S1$. The subscript $H1$ indicates that the key used is that which is shared between $H1$ and the CryptoApp.

Step 2: $S1$ like before, forwards a PacketIn request to the controller.

Step 3: The controller can still look at the interface $S1$ received the packet on, to uncover $H1$ as the original sender. It then forwards this information, along with the header, to the CryptoApp, which uses its lookup table to decrypt the source-destination and uncovers $H9$ as the intended destination. The CryptoApp then uses the key it shares with $H9$ to re-encrypt $h1\_ip$ and $h9\_ip$ to $enc\_h1\_ip_{H9}$ and $enc\_h9\_ip_{H9}$ respectively. It returns the plaintext and re-encrypted addresses back to the controller, which once again, computes a path and installs the relevant flow instructions on $S1$, $S2$ and $S3$.

Steps 4,5,6: These steps are identical to before albeit we now only have a single rewriter node. Note that the rewrite at $S1$ ensures that the source and destination are now encrypted with $H9$'s key.

The packet $H9$ receives has been encrypted with its own shared key and thus, it can decrypt and uncover the original packet. As before, the next packet will be handled by the installed flow rules without requiring the controller.

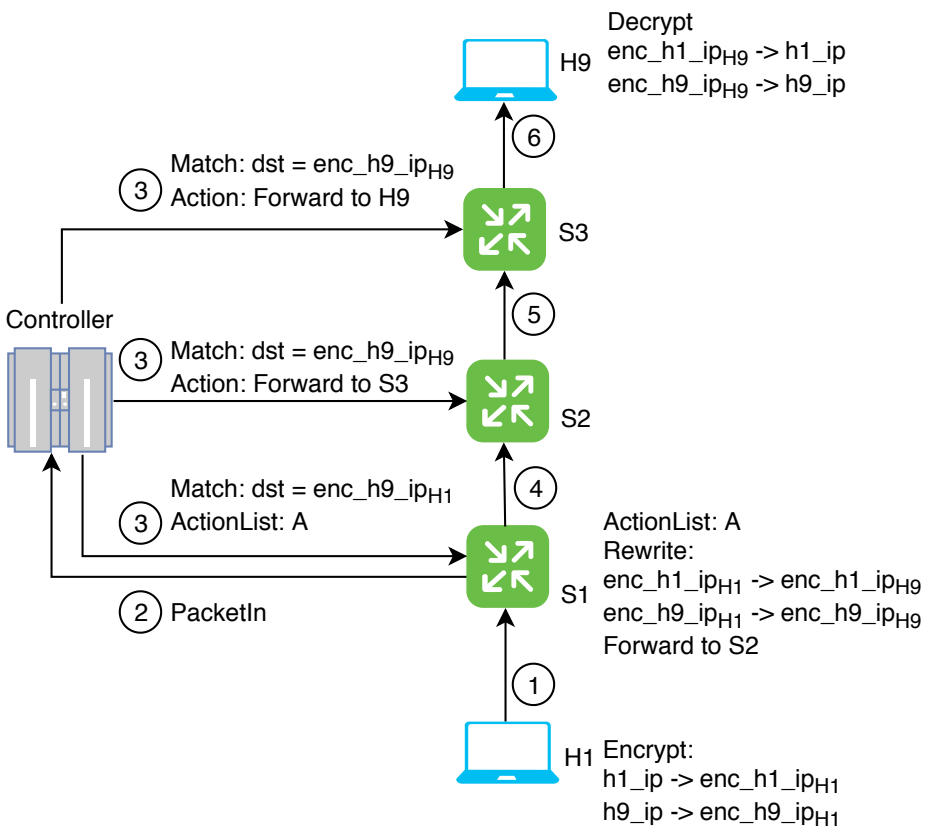


Figure 5.2: Client communication in solution for improved anonymity

5.2.3 Discussion

A few aspects of this design stand out. Primarily, this solution will require modifications on both the network and client side to enable host-controller key exchange. However, we argue that the associated cost is minimal as the changes effected are minor and do not involve any new architectural components. Thus, we continue to meet our objective **O4**. Moreover, we now have the obvious benefit of obfuscating the network layer addresses from the edge FEs. In this design, every FE only ever observes ciphertext addresses, encrypted with the keys of either the source or destination client. Thus, we now truly achieve **O1**.

This comes at the cost of requiring 3 more cryptographic operations than before. However, like our first solution, this is only required on the first packet of a flow and will thus, result in just a constant factor increase in overhead. Therefore, we can continue to maintain our claims to **O3**. We recall that to bootstrap communication, hosts need to communicate with the CryptoApp application over the standard routing network. However, during initialization, the hosts and the CryptoApp do not have any shared keys and thus can only communicate using regular non-privacy protected traffic (though it can still be protected with higher level protocols like TLS). Thus it is critical that we support TrafFu's ability to simultaneously support regular and privacy-protected traffic as it enables this bootstrapping.

However, we still carry the problem of an injective obfuscation scheme, which means we once again fail to achieve **O2**. This brings us to our final design that takes the same approach to achieve **O1**, but looks to prevent a one-to-one plaintext to ciphertext mapping in order to achieve **O2**.

5.3 Design for resilience against Traffic Analysis

5.3.1 Network Model

We borrow the same model as our previous design and make one small modification. When the client joins the network, it shares a master key with the CryptoApp and both parties use it to generate some m number of shared keys.

5.3.2 Packet Forwarding Process

The packet forwarding process is virtually identical to our second solution. The only difference is that the client can randomly select any of the shared keys with which to encrypt the packet. To better illustrate, we can now represent the encrypted address as $enc\_hk\_ip_{Hk_i}$, where i represents the ith, or index of the, shared key between host Hk and the controller.

In this design, packets are only routed to the controller the first time a client encrypts the source-destination of a flow with key i. Note that different flows encrypted with the same key, will still generate different ciphertexts and will have to be forwarded to the controller. Similarly, the same flow encrypted with a different key, will also generate a different ciphertext and will have to be forwarded to the controller. This gives us a tradeoff between anonymity and overhead and is discussed in greater detail in Chapter 6.2 and evaluated in Chapter 8.

5.3.3 Discussion

By sharing more than one key, we need some way for the client and controller to communicate which key (key index) is being used. While we leave the minor details to implementation specifics, we make a similar assumption to LAP [28] by reasoning that the IP header can be used to embed this information. [2] We note that the amount of information leaked by embedding the key index into the packet header is minimal. If two packets are observed with the same encrypted source-destinations, the adversary is already reasonably confident that the packets belong to the same flow. IPv4 addresses have 32 bits of entropy and as such, the chance of ciphertext collision is already fairly low. IPV6 addresses have even more entropy. Therefore, while adversarial confidence will increase if the ciphertexts and the embedded indices of the two packets are observed to be the same, the degree of increase will be minimal.

Like our previous solution, network layer addresses are protected against all FEs, thus maintaining our claims to **O1**. However, in using multiple keys to encrypt packet headers of the same flow, we finally provide a non-injective obfuscation scheme. So long as there are other privacy-protected flows passing through the compromised FE, the attacker will not, without probability better than pure guessing, be able to infer a relationship between two packets with different ciphertexts. We can now claim to have successfully raised the cost of conducting traffic analysis attacks, thus achieving **O2**. We discuss the specifics of **O1** and **O2** in greater detail in Chap 6. We make no changes to the infrastructure setup in our previous design and can continue to claim to achieve **O4**.

While this design achieves our stated objectives, it comes at a cost. In an injective encryption scheme, all packets of a flow are handled by a single flow table entry. However, in this non-injective scheme,

[2] There are several approaches to incorporate the index of the key used into the IP packet header. A few potential approaches include adding an additional options field for TrafFu or using the TOS+ECN fields when permissible.

every flow generates m flow table entries, where m is the number of keys used to encrypt packets of that flow. We discuss this further and propose a potential mitigation in the following seection.

5.4 Aggregatable Encryption

In SDNs, FEs make routing decisions based on rules contained in flow tables. Packets that do not match any table entry are forwarded to the controller which can add, update and delete rules in the table. Generally, flow tables require significantly less memory than traditional Ethernet switches, because FEs only need to maintain the state of flows in progress, rather than remember all the flows they are likely to encounter [8]. However, these tables use Ternary Content Addressable Memory (TCAM), which by virtue of being power hungry and expensive, is typically limited [44].

As we alluded to earlier, the final design of TrafFu can add significant stress to the flow tables. As mentioned earlier, every packet in a flow that is encrypted with a key for the first time, is forwarded to the controller, which then installs a new flow table rule for every FE on the path of the flow. In a situation where the number of keys being used in the system is large, network FEs can suffer as their flow tables can fill up rapidly, causing *flow congestion*. Intuitively, the number of clients in the network, the number of keys used per client, and the number of flows a client is involved in are all directly proportional to the number of ciphertexts generated. In large scale networks, with a significant fraction of clients seeking anonymity, the FEs may suffer from flow congestion relatively quickly.

Traditionally, congestion is offset by *flow aggregation*, where topologically related flows are handled by a single rule. For example, packets from two different flows may share common path segments as they travel to their destinations. An intermediate node common to both, will route them identically. Subsequently, the controller can save space on the flow table of the intermediate node, by programming a single rule for any packet destined to that subnet.

We can extend TrafFu to incorporate flow aggregation in order to reduce flow table utilization, while still providing some measure of privacy protection. We exploit the fact that routing is hierarchical, and so packets that share the same destination subnet, tend to take similar paths and thus, may share flow table entries on FEs along the shared portion of the path. To expose this sharing to those routers, we extend TrafFu to encrypt the upper octets of the address (either /8 or /16 depending on configuration) with the same key across all packets, determined by the controller, while the lower octets are still encrypted with different, host-specific, keys. Thus, all packets bound for the same subnet will have the same upper octets in their destination addresses. For example, two packets bound for the addresses $8.a.b.c$ and $8.x.y.z$, the common subnet 8.0.0.0 would be encrypted identically while the lower address bits would be encrypted with the host-specific keys. I.e., if 8.0.0.0/8 is encrypted to 42.0.0.0/8, then $8.a.b.c \rightarrow 42.a'.b'.c'$ and $8.x.y.z \rightarrow 42.x'.y'.z'$. Since the host does not know the key used to encrypt the upper octet, one of the nodes (we can imagine the rewriter node facing the destination), will be tasked with rewriting the encrypted upper octet back to its original value. This allows FEs to aggregate on the subnet, while still not revealing the true destination of the host. We highlight this process in Figure 5.3, which shows how packets destined to both 8.1.1.1 and 8.1.1.2 are handled by a single rule on all intermediate (non-rewriter) nodes, $S2$ and $S3$. Recall from our our design proposed in Section 5.2 that $x.y.z$ and $x'.y'.z'$ are the host identifier bits encrypted with $H1$ and $H3$'s keys respectively. Similarly for $a.b.c$ and $a'.b'.c'$. Note that for the sake of brevity, we leave out the controller and its interactions with the switches from the figure. We evaluate the additional flow table utilization caused by TrafFu, as well as the effects of

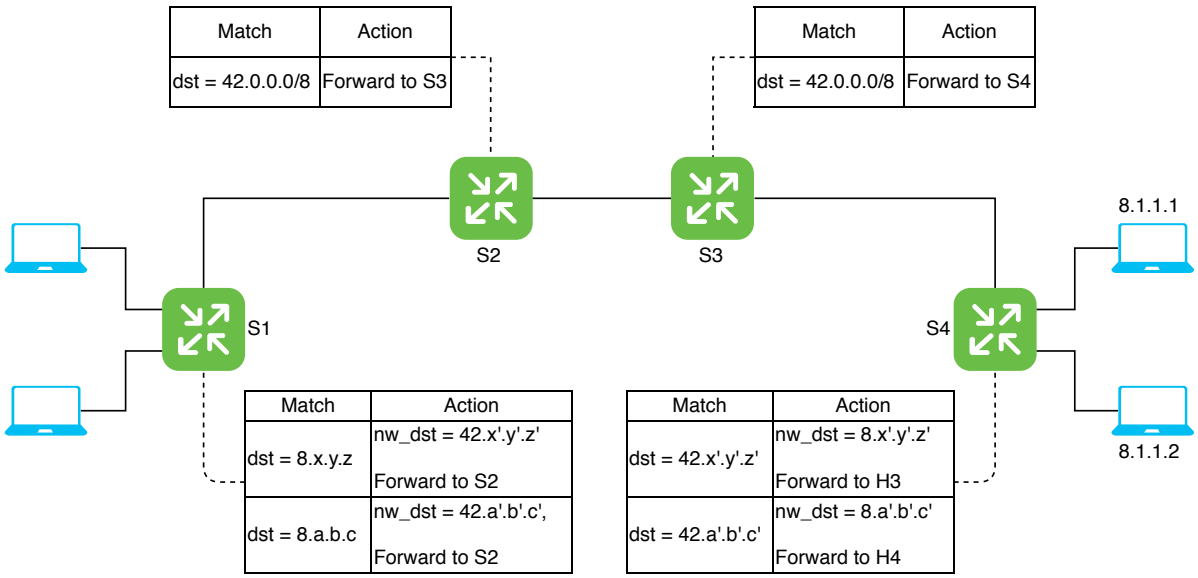


Figure 5.3: Illustration of the reduction in switch Flow Table entries when using aggregation

flow aggregation in Chapter 8.

Compared to TrafFu without encryptable aggregation, this allows the attacker to know if different packets are bound for the same subnet and will reduce the space the attacker has to search if they are trying to reassemble encrypted packets into flows for traffic analysis. We analyze this security-resource trade-off in Chapter 6.2.

5.5 Reverse Ciphertext Lookup Table (RCLT)

In all of our designs, we assume that if a PacketIn request arrives from an FE, the controller can look at the interface that the FE received the packet on to uncover the original source. However, this is only true if the request arrives from the first hop node. In situations wherein a flow table rule expires, or if an intermediate FE is replaced in real time, PacketIn requests can arrive from non-first hop nodes as well. In these cases, the interface only informs the controller of the previous hop of the packet. To account for these situations, when the controller first sets up flow rules on FEs, it maintains a mapping of the encrypted-to-plaintext source-destination addresses in a table we call the *Reverse Ciphertext Lookup Table* (RCLT). Subsequently, when a PacketIn request does arrive from a non-first hop FE, the controller can consult the RCLT, uncover the original end-points of the connection and handle it appropriately as before.

5.6 Payload encryption

As we mentioned earlier, TrafFu is designed solely to restrict the ability of an adversary to infer useful information from network layer packet headers. We assume the fact that higher level protocols are protected using current best-practices and do not leak information. However, it is worth pointing out higher level protocols can be a veritable treasure-trove of information and if unprotected, can undermine TrafFu's best efforts at hiding network layer packet headers. For example, consider the case where the

sender and receiver use a protocol like TLS to establish a secret key over TrafFu. In order to avoid man-in-the-middle attacks, this protocol involves a certificate exchange between both parties. However, at least one of the certificates exchanged must be in plaintext and as such, will reveal information about the certificate owner. An adversary that encounters a certificate bearing packet can trivially defeat **O1** without even having to look at network layer headers.

We note that there are existing measures that aim to encrypt client side certificates such that at most, the identity of only the server is revealed [65]. Encrypting server-side certificates however, will require additional functionality from the network provider, such as an IPSec network tunnel setup over a flow's path. This will encrypt all payloads as they enter the network and decrypt them as they leave, while the headers are protected by TrafFu.

Chapter 6

Security Analysis

In the following section, we evaluate TrafFu's resilience against a variety of attack vectors. We begin by measuring the degree of sender and receiver anonymity (**O1**) against an adversary located at different points in the network. We then look to evaluate the increase in cost for an attacker looking to conduct traffic analysis attacks (**O2**). We augment this section by considering the tradeoff between anonymity and security before also discussing some other attacks and their efficacy against TrafFu.

6.1 Sender and Receiver Anonymity

In this section, we discuss the relative levels of anonymity for a user against an adversary located at different points in the network. As mentioned earlier, we define sender anonymity as the ability of the sender of a message to be unidentifiable within an anonymity set. Receiver anonymity is similarly defined from the perspective of the receiver. We declared our objective **O1** as achieving sender anonymity on all nodes except for the first hop node and receiver anonymity on all nodes except for the last hop node. To evaluate this metric in TrafFu, we take the view of an adversary that has compromised the following nodes

- **Edge node:** Recall that we define an edge node as a terminal node (first or last hop) along a flow's path. Due to the physical interconnections between edge nodes and clients (i.e., our source or destination), a first hop node always knows the sender of the packet while the last hop node always knows the receiver of the packet. However, for each node, the other end-point is encrypted and therefore, the first hop node is unaware of the recipient while the last hop node is unaware of the sender. Hence, we achieve our stated objective **O1** in this scenario.

- **Intermediate node:** We define an intermediate node as any *non-edge* node, or one that lies between the first and last hop nodes of a path. Packets traversing such FEs have their addresses obfuscated and thus, do not directly expose the sender or receiver to these nodes. This helps us achieve **O1**. Sender/receiver anonymity on intermediate nodes comes with the caveat of assuming an adversary that is unable to leverage topological information. However, given that TrafFu can only generate paths depending on the physical interconnections between different FEs, it is possible for an adversary to gain some amount of topological knowledge. This reduces the anonymity set for a packet in a compromised node to all hosts that could potentially be involved in a flow that routes through the compromised node.

We note that it is possible for an adversary to compromise multiple nodes in a network as well. In this case, the set of possible flow end-points is the Cartesian product of the sender anonymity set intersection and the receiver anonymity set intersection across all compromised nodes. An adversary that manages to compromise the first and last hop nodes of a flow can trivially uncover the connection endpoints. However, this only gives the adversary information limited to a single flow and is unlikely to satisfy an attacker looking to conduct large scale surveillance against a network.

In comparison to related network works like LAP, Dovetail and PHI [28, 54, 12], TrafFu offers similar guarantees over sender anonymity but offers stronger receiver anonymity guarantees. This is primarily due to the fact that TrafFu outsources routing decisions away from the FEs and onto the controller. As a result, there is no need for any FE to ever observe the destination in plaintext

Another drawback of some network anonymity protocols [28, 54] is that they reveal some information about a flow's path, such as the total length and in some cases, the position of an FE on the path. This type of information significantly reduces the anonymity set for a user [11]. In comparison, TrafFu leaks no information about the path length or the position of a node on it.

6.2 Attacks on Anonymity

Traffic Analysis: Traffic analysis is an attack where an adversary takes in network traces as input and based on patterns observed in the traces, makes inferences about information embedded in the traffic [14]. This can range from the type and frequency of websites being visited, to the actions performed on a smartphone to the operating system of the device involved. There are many variants of this attack, that have proved successful [61, 27, 50, 49, 72, 19, 59]. However, as noted by Juarez et al. [31], a critical assumption in these attacks is the ability for an attacker to collect a well-defined traffic trace, or in other words, isolate all packets belonging to a target flow. In related works on network anonymity [18, 28, 11, 54, 12], a flow has to first set up a dedicated path, through which all packets in that flow must pass. This means that headers for packets in the same flow must have unique identifiers i.e., be injective. Therefore, an adversary can easily isolate all packets belonging to a target session, even if there exist multiple flows simultaneously interspersed.

In TrafFu, there is no requirement for a flow to set up and follow a fixed path, and consequently, no requirement for a session identifier. This allows us to use a non-injective obfuscation scheme and encrypt packet headers with different keys. Thus, the adversary can now confidently isolate only packets with the same ciphertext addresses. Assuming that TrafFu uses an infinite number of keys, we can make traffic analysis even harder by making every packet's network layer headers cryptographically indistinguishable. The adversary has no trivial way of isolating packets belonging to a single flow. More realistically however, with a finite set of keys, it reduces the number of packets that can be linked to a flow, to a function of the number of keys used.

Let us take the perspective of an adversary that has compromised a router and is looking to conduct large scale traffic analysis attacks on the flows it observes. We imagine n flows, of p packets each, passing through this node, with each flow being encrypted with m different keys. The adversary's goal is to isolate a certain fraction f of packets belonging to a single flow. In order to do so, the adversary would have to isolate $t = f * p$ packets from a message pool of $n * p$ packets. We make an additional assumption by equipping the adversary with the ability to perform traffic analysis on any t packets, not necessarily sequential. In other words, the adversary need not get packets $i, i+1, i+2, ..., i+t$, but simply requires

any t of the p packets. This simplifying assumption actually makes the adversary stronger as typically, traffic analysis relies on inter-packet timings, for which the adversary would need sequential packets. Packets with differently encrypted source-destinations are essentially indistinguishable from one another and consequently, the attacker has no obvious way of knowing if they belong to the same flow. However, assuming no collisions, if $p > m$, then in a single flow, m groups of p/m packets each will have the same ciphertext source-destinations, and as such, can be linked to the same flow. Therefore, the pool of $n * p$ packets can be broken down into $n * m$ groups. In order to obtain t packets from these set of groups, we need to choose the right fraction of them, i.e. $f * m$ groups. Therefore, the overall task of the attacker can be described as searching the combination:

$$\binom{n * m}{f * m} \tag{6.1}$$

If we consider a scenario of a router that's carrying a large sampling of network traffic, transiting $n = 1000$ flows, of $p = 128$ packets each, wherein each client is using $m = 4$ keys and an adversary would like to extract $f = 0.5$, or 50% of the traffic belonging to a target flow, she would have to perform traffic analysis 8 million times (i.e. $\binom{1000*4}{0.5*4}$) raising the cost to use traffic analysis by the same factor.

If encryptable aggregation is in use, this reduces the space an attacker must search as she need only search the packets that have the same encrypted address prefix. This reduces the search space by some factor r so that the number of combinations the attacker must try is now:

$$\binom{\frac{n*m}{r}}{f * m} \tag{6.2}$$

The value of r is heavily dependent on topology and traffic patterns and we empirically determine r to be on the order of ˜20 in our experiments, described in Chapter 8, which still requires the adversary to do roughly 400K traffic analysis operations. Given this situation, we note that side channel attacks become more effective [75, 43, 9, 23, 24, 30], but these require attackers undertaking targeted deanonymization, which are outside of our attack model.

Security vs Overhead tradeoff We recall from Chapters 5.2 and 5.3 that every time a message goes to the controller, there are 4 cryptographic operations involved (1 encryption by the source, 1 decryption by the CryptoApp, 1 encryption by the CryptoApp and 1 decryption by the destination), while n additional messages are generated (instructions by the controller) and n additional flow rules are installed, where n is the number of FEs on the path. For messages that do not go to the controller and are handled directly by the FEs, only 2 cryptographic operations are required (1 encryption by the source and 1 decryption by the destination) and no other additional messages or flow rules are generated. If we assume that flow rules on FEs don't expire, the extra load on the controller/CryptoApp and the additional latency, bandwidth and flow table overhead are incurred every time a new ciphertext source-destination is encountered by the FEs, or in other words, a key is used for the first time in a particular flow. If the number of keys used is small, this extra load and network overhead are minimized. However, this makes it easier to carry out traffic analysis as the ciphertext source-destinations can act as a session identifier. Thus, an adversary will be able to isolate these packets belonging to the same flow. The adversary can then conduct traffic analysis, thus defeating **O1** and **O2**. At the other extreme, if the number of keys used is large, the load on the controller/CryptoApp and FE flow tables increases, which harms (**O3**), but it makes traffic analysis harder to execute because the adversary would be able to

isolate a fewer number of packets belonging to the target flow. We evaluate this tradeoff in Chapter 8.

Switch Identity Forging Attack: An integral piece to TrafFu's security guarantees is that each FE knows only of the previous and next hop for a particular packet. However, if an FE is then able to impersonate the next hop FE and once again send a PacketIn request to the controller, it can find out the next-next hop, and so on. In TrafFu, every FE maintains a TLS session with the controller. This allows the controller to verify the authenticity and integrity of any message received by an FE, in addition to the identity of the FE itself. This takes away the ability of an FE to impersonate its counterparts, thus defeating this attack.

Session Linkage: Given that there are no dedicated circuit setup packets, without getting into protocol semantics, there is no trivial way of identifying the first packet of the session even if no encrypting keys are used. When we increase the number of encryption keys used, this problem is made even harder as it is difficult for an adversary to collect all packets belonging to a session to begin with, thus making it hard to know which packets to link.

Response Timing Attack: As mentioned earlier, knowledge of a node's position on a path can significantly degrade the anonymity set. To that end, a node forwarding a packet can measure the time taken to receive a response to infer some knowledge about its position on the path. TrafFu is not vulnerable to this attack because *forward* and *backward* paths of a connection are encrypted independent of one another. As a result, a FE has no way of knowing if the second packet it receives is actually a reply to the originally forwarded packet, and the cost of searching for such matches can be modelled in a way similar to the traffic analysis attack. Packets on each path are encrypted using a different set of keys and as such, even reusing the same key will only increase the adversary's knowledge of one of the paths, without compromising the other. Additionally, the added benefit of setting up independent forward and backward paths is that they need not traverse through the same route. If the controller deems such an attack a threat, it could viably alter its routing policy to use different forward and backward paths.

Replay Attacks: A replayed packet will simply traverse the same path as the original packet and be duly handled by the intended recipient, thus giving the attacker no new information.

Path Modification Attack: In a situation where there are colluding adversaries, but only one of them lie on the path being traversed by a target flow, the adversary may deliberately misroute the packet to the other adversary, causing it to generate a PacketIn request and hopefully, from the adversary's standpoint, a response with routing instructions. To counter this attack, while processing each PacketIn request, the controller should check if the router sending the request lies on the expected path for the target flow. Any deviation from the expected routing policy will inform the controller of unusual activity, identifying the router as potentially compromised or misconfigured.

Payload/Content Analysis Attack: As we alluded to earlier in Section 5.6, depending on the application or protocol used, the payload may contain data that can deanonymize a flow. While future instances of TrafFu could be extended to offer payload protection, in our current proof-of-concept, we expect the user to employ higher-level protocols to protect such data.

6.3 Other Attacks

Denial-of-Service (DoS) Resilience A malicious FE may try to bombard the controller with PacketIn requests and subject it to a computational DoS attack. However, given that the controller is the brains of the network, it is not that hard for it to realize that it is receiving repeated requests from the same node(s) and subsequently proceed to disconnect from it, while simultaneously invalidating all paths that traverse through it. Moreover, compared to proposals that require FEs to perform cryptographic operations, TrafFu will require an adversary to consume a lot more resources to cause a computational DoS attack on an FE.

Chapter 7

Implementation

In this section, we discuss our implementation of the various components that comprise of TrafFu and outline the various Open Source tools used in the development of our prototype.

7.1 Network

We use Mininet [36] to create a network topology of interconnected switches which are based on Open vSwitch and support OpenFlow protocol 1.0 [40]. By allowing us to develop an actual prototype, Mininet pushes our design to beyond the realms of pure simulation. Mininet employs lightweight OS virtualization features such as process groups, network namespaces and virtual Ethernet pairs to create containers for individual network components. This allows for faster speeds and for the system to scale to more hosts.

Network links are implemented using virtual Ethernet (veth) pairs that appear as a wire connecting two virtual interfaces. Each interface appears as a fully functional Ethernet port to all system and application software. Linux Traffic Control (tc) [2], a utility program used to configure the Linux kernel packet scheduler, is used to control the data rate of each link.

Hosts are a group of processes moved into their own network namespace. This allows them exclusive access of networking artefacts (interfaces, ports etc) in their namespace. Each host connects to the network via its own virtual Ethernet interface. Mininet uses CPU bandwidth limiting to control the CPU fraction available to each process group.

Mininet typically uses the default Linux bridge or Open vSwitch running in kernel mode to switch packets across interfaces. Software OpenFlow switches provide the same packet-delivery semantics that would be provided by a hardware switch. While both user-space and kernel-space switches are available, for performance reasons, we opted for the latter.

Mininet networks can be configured to a controller that can run anywhere as long as it has IP-level connectivity with the switches. We discuss our controller implementation in the following section.

7.2 Controller and CryptoApp

We implement the controller as an L3 Learning Switch using Pox [47], a Python-based open source controller. On startup, the module sets up a TLS connection with every switch in the network. We

further make use of a prepackaged *Discovery* class, that causes OpenFlow switches to send out specially crafted LLDP packets. These packets raise Link Events when a link goes up or down. We extend the Discovery class to extract information from these events and maintain an up to date knowledge of the topology. The module also handles PacketIn requests and forwards the necessary packets to the CryptoApp. In order to correctly encrypt/decrypt the packet, we use the TOS field in the IP header to embed the index of the key used. A default value in this field indicates an unencrypted packet, which is not forwarded to the CryptoApp. Lastly, the controller also maintains the RCLT as a dictionary, to store the ciphertext to plaintext source-destination mapping.

We developed the CryptoApp as a separate module that is instantiated within the controller. The module establishes a shared key with every requesting client. It then generates the per-host array of symmetric keys using HKDF [34], a simple HMAC based Key Derivation Function. We use SHA-256 as the HMAC's Hash Function. For encryption, we use Python's Cryptography Library, PyCrypto [38].

7.3 Client

Mininet hosts were a convenient abstraction for our clients. Within every host, we ran a network facing proxy, that on startup, established a shared key with the CryptoApp. The proxy could then decrypt packets as they exited the hosts' network namespace and vice versa. We used *Libtins* [21], a high-level, multiplatform C++ library, to act as our proxy and used *CryptoPP* for our cryptographic operations.

Chapter 8

Evaluation

We mentioned previously that the main cost of TrafFu is that it results in a noticeable increase in flow table utilization. We begin this section by evaluating the degree of this increase as a tradeoff with privacy. We then proceed to compare TrafFu against related works for various sources of overhead.

A typical evaluation metric for most research in the networking spectrum is latency and throughput. These metrics are typically measured over design implementations on real hardware, against real-world topologies and real-world traffic traces. TrafFu is a proof-of-concept and its implementation on real hardware is left for future work. As such, we do not obtain values for latency and throughput. However, we do not believe that TrafFu imposes any major costs in this regard. Our design uses unmodified, commodity SDN FEs. Consequently, there should be no impact to the speed or latency of packet forwarding. Further, while TrafFu occasionally forwards a packet to the controller, any additional latency due to switch-controller round trip times and an increased latency on the controller is minimal as we expect switch-controller traffic to be strongly dominated by regular switch-switch packet forwarding traffic. In the remaining sections, we discuss and show how TrafFu is an improvement on previous designs in this regard.

8.1 Flow Table Utilization vs Privacy Tradeoff

8.1.1 Experimental Setup

In order to quantitatively measure the effects of our design, we measured the average increase in flow table entry (FTE) utilization as a result of TrafFu. Using our implementation as outlined in Chapter 7, we took readings for this metric across two topologies, the real-world backbone network Abilene (Figure 8.1a) and a standard datacenter fat-tree (Figure 8.2a). In the Abilene network, we connected each switch to 10 end-hosts (110 overall). The fat-tree network was configured with 6 pods, which resulted in 3 end-hosts (54 overall) connected to each edge switch. In both cases, all the hosts attached to a particular switch were part of a unique subnet. We made each host engage in bidirectional flows with two other hosts, and used IPerf as our synthetic traffic generator to send 2000, 512 Byte packets across each flow. Figures 8.1b and 8.2b highlight the increase in FTE usage as a result of TrafFu. We further augment the graph with the same metric, but employing flow aggregation as defined in Section 5.4. Lastly, in order to understand the relationship between FTE usage and anonymity, we simultaneously plotted the search space for an attacker looking to conduct traffic analysis (TA) attacks. We calculated

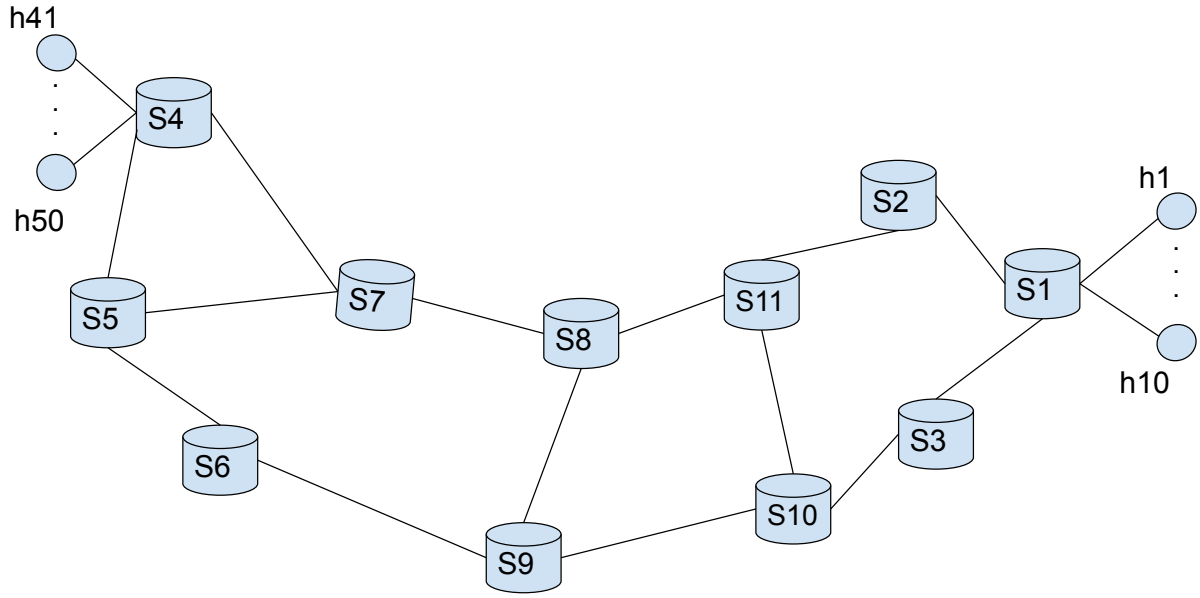
these numbers using Equations 6.1 and 6.2 as defined in Chapter 6.2.

In order to quantitatively measure the effects of our design, we deployed Mininet in a Ubuntu 16.04.5 VirtualBox instance to simulate two topologies, the real-world backbone network Abilene (Figure 8.1a) and a datacenter fat-tree (Figure 8.2a). In the Abilene network, we connected each switch to 10 end-hosts (110 overall), while the fat-tree network was configured with 6 pods, which resulted in 3 end-hosts (54 overall) connected to each edge switch. In both cases, all the hosts attached to a particular switch were assigned a unique subnet. We made each end-host engage in bidirectional flows with two other hosts, and used IPerf as our synthetic traffic generator to send 2000, 512 Byte packets across each flow. We plot the average increase in flow table entry (FTE) utilization vs a baseline of regular network traffic (TrafFu disabled) in Figures 8.1b and 8.2b. We also compare with and without aggregatable encryption with a /24 subnet. On the same graph, we also depict the search space for an attacker to mount a website fingerprinting (WF) attack. We calculated these numbers using Equations 6.1 and 6.2 as defined in Chapter 6.2. From our experiments, we empirically determined the average number of flows (n) passing through a switch to be 60 and 42 for each topology respectively, while we estimated the reduction factor for aggregation, r, to be 20. We also conservatively empowered the attacker to successfully conduct the attack if she extracts $f = 0.5$, or 50% of all packets belonging to a flow, in any sequence.

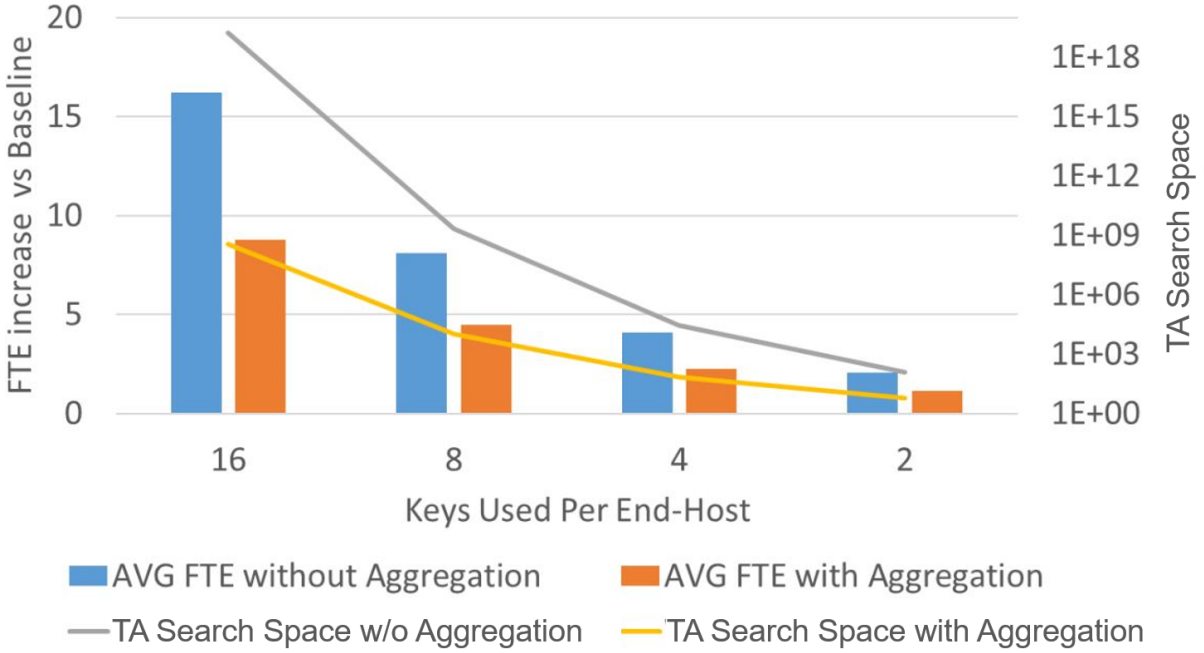
8.1.2 Discussion

Encrypting each flow with m keys effectively creates m flow entries per flow. This is confirmed by the linear relationship between the increase in flow table utilization (without aggregation) and m. We also see that aggregation consistently reduces flow table utilization by roughly 50%. This is a rather interesting observation. To make sense of this, we refer back to Figure 5.3, which shows that flow table space can be saved by handling multiple flows to the same subnet with single rule. It is important to note however, that flow table utilization remains the same on the first and last hop (edge) nodes as there still needs to be one rule for every encrypted source-destination address. The real benefit of flow aggregation is felt on the intermediate nodes. Without aggregation, every rule installed at an edge node, would need a corresponding rule at an intermediate node. This can also be seen in Figure 5.2. With aggregation, an intermediate node only needs a single rule for every subnet that routes through it. Thus the reduction of flow table entries in the network is proportional to the average number of intermediate nodes across all flows.

Let us take a simple example of 2 subnets, both of which consist of 5 clients each. Any path between the two subnets is 10 hops long. We assume that clients from one of the subnets pick a single client from the other subnet to engage in a unidirectional flow. This gives us a total of 5 flows. Further, we assume that each client uses 4 keys for its obfuscation purposes. Based on our understanding so far, this means that without aggregation, each FE will have $5 * 4$ or 20 flow table entries — a single flow rule for each key used in each flow. This gives us a total of 200 flow rules in the network. Since each flow is travelling between the same subnets, the intermediate nodes can employ aggregation and use a single rule to handle them. Thus, while the 2 edge (rewriter) nodes will still have 20 flow rules each, the 8 intermediate nodes will have 1 flow rule each. This gives us a total of $2 * 20 + 8 * 1$ or 48 flow rules. This is a reduction factor of around 4.16 or 76%. Clearly, the reduction factor is proportional to the number of intermediate nodes, or in other words, the length of the path between the two subnets. The longer this path is, the greater the number of intermediate nodes and therefore, the greater the reduction in flow table utilization.

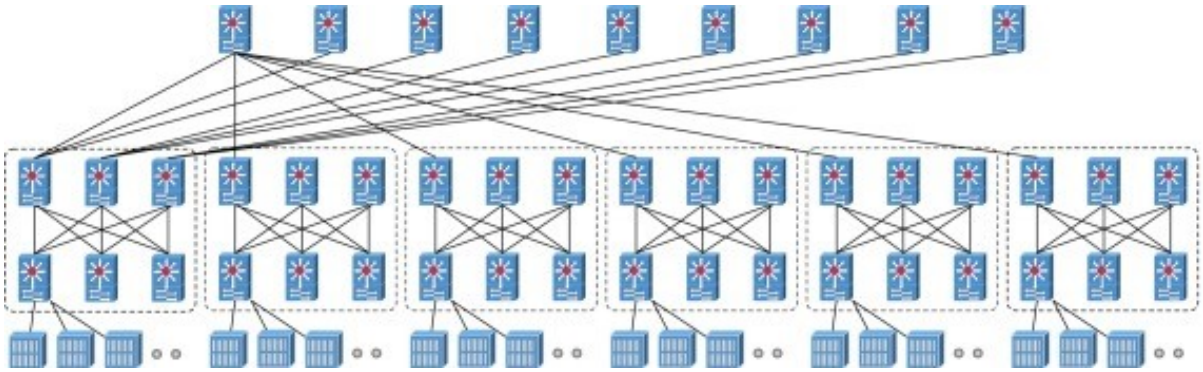


(a) Abilene Topology with 10 end-hosts on each switch

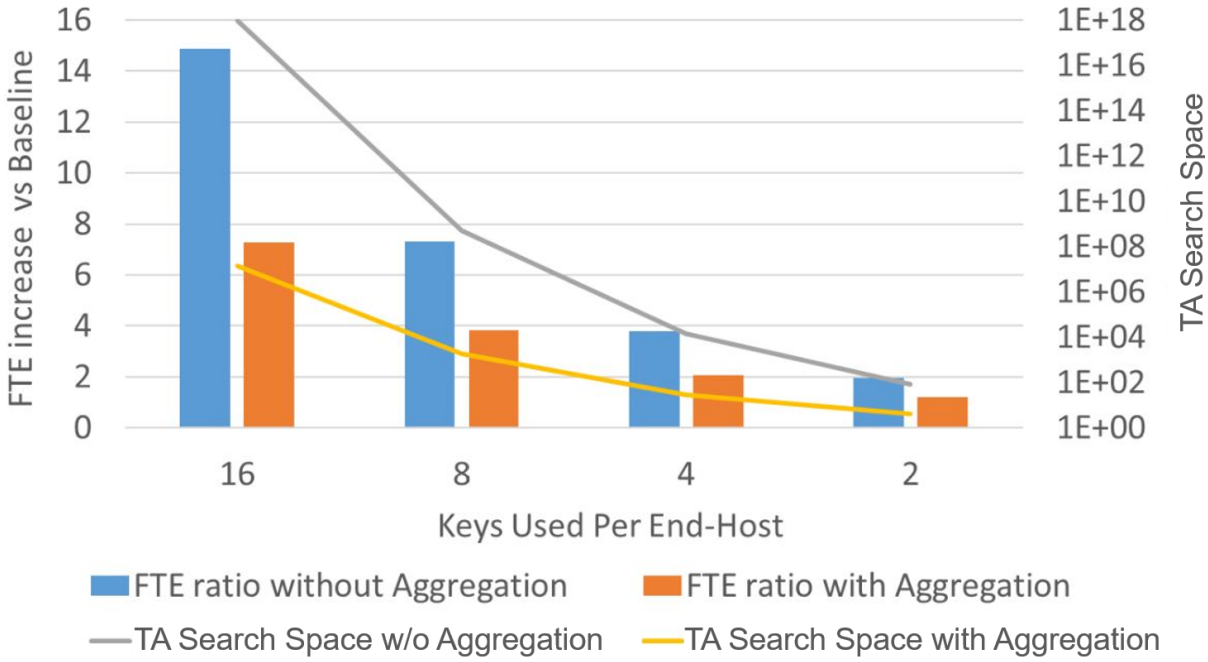


(b) Average number of Flow Table Entries (FTEs) varying the number of keys generated in Abilene Topology. The figure also charts the search space for a Traffic Analysis (TA) attacker

Figure 8.1: Privacy vs Overhead tradeoff on the Abilene topology



(a) Fat Tree Topology with Pods k=6 as taken from [41]



(b) Average number of Flow Table Entries (FTEs) varying the number of keys in the Fat Tree Topology. The figure also charts the search space for a Traffic Analysis (TA) attacker

Figure 8.2: Privacy vs Overhead tradeoff on the Fat tree topology

Coming back to our experiment, the 50% reduction in flow table utilization hints at the fact that on average, we generate flows of short path lengths. We expect that flows in real-world topologies should, on average, produce longer paths and magnify the effect of aggregation, thus making TrafFu scalable to larger networks.

A downside however, is that aggregation also reduces resistance to traffic analysis by roughly a square root (note the log scale). While this might paint a bleak picture for TrafFu to realistically defend against traffic analysis attacks, there are a few caveats to our experiments.

We note that the effort required for traffic analysis is directly proportional to the number of flows (m) and the number of keys (k). As pointed out earlier, we estimated that the average number of flows passing through an FE per topology was 60 or 42 respectively. Real-world FEs, particularly those with access to a large sampling of network traffic (and therefore the likely targets of mass surveillance attacks), will likely have a far greater value of m. Moreover, as mentioned above, longer paths in real-world topologies will amplify the benefits of aggregation and potentially allow for TrafFu to increase the number of keys user per client. The combination of an increase in m and an increase in k means that while aggregation does reduce the search space for traffic analysis by a square root, in absolute terms, it will still increase the cost of such attacks considerably.

8.2 Evaluation of Overhead

This section looks at some related works and pits them against TrafFu, comparing for various sources of overhead.

8.2.1 Latency and Computational overhead

A common theme of other anonymity protocols [18, 54, 28, 12, 11] is that they all require a high latency path setup phase in addition to increasing the computational overhead on the FEs and end-hosts via a large number of cryptographic operations.

Tor's *telescoping* circuit construction approach requires $\mathcal{O}(n^2)$ cryptographic operations and $\mathcal{O}(n^2)$ number of messages, where n is the number of nodes on the path. On the forward path of every message, the source has to perform $\mathcal{O}(n)$ consecutive encryptions to create the *onion*, while every on-path node performs a single decryption to 'peel' off a single layer before forwarding.

In **Hornet**, a dedicated circuit setup packet is passed to every node on the path, which generates and encrypts a key into the packet. Once the circuit is setup, the first packet sent to the destination contains information about the backward path i.e., from the destination to the source, and is another source of overhead. On every regular message transfer, like Tor, the source has to onion encrypt the payload. Each on-path node however, has to perform two cryptographic operations — one to extract the packet state, and one to remove/add a layer of the onion.

Similarly, in **LAP**, each on-path node encrypts its routing decision into a segment of the packet header during circuit setup and decrypts this segment during regular message transfer.

Dovetail has a much more involved circuit setup scheme. Instead of directly routing to the destination,

Scheme	Header Length
Tor	3+11.r
LAP	12+2s.r
Dovetail	12+2s.r
PHI	>12+2s.r
HORNET	8+3r.s
TrafFu	20

Table 8.1: Header Length for different schemes as per analysis done in [11]. Here s is the length of the symmetric key and r is the maximum path length

the source first contacts a *matchmaker* node and sends it the address of the destination and another *dovetail* node, encrypted using the matchmaker node's public key. Once decrypted, the matchmaker computes a list of possible paths from the matchmaker to the destination that passes through the dovetail and sends this list to the source. The source then picks an option and conveys it to the matchmaker, which then proceeds to build the rest of the circuit. Like LAP, each on-path node encrypts its routing decision into the packet header.

PHI is also like Dovetail in that it uses 2 indirection nodes, the *helper* and *midway* nodes. The source encrypts the destination address using the helper node's public key. The helper node uncovers the destination and goes backwards along the path until it finds a midway node willing to route to the destination. Once again, each on-path node encrypts its routing decision into the packet header. However, for the benefit of greater anonymity guarantees, PHI's circuit setup phase can sometimes fail and may require a few retries before success.

In the case of **TrafFu**, we reduce the computational load on FEs as we do not require any cryptographic operations from them. That burden is pushed onto the CryptoApp, which as an SDN App, can be designed to handle a large computational load. Additionally, there are no public key cryptographic operations at all. Moreover, we do not require a single circuit setup packet to be passed around to every FE hop-by-hop. Once the controller determines the required flow rules, it can install them on the necessary FEs in parallel thus reducing the path setup latency (**O3**).

8.2.2 Path Length

In a system like Tor, circuits are typically 3 Tor nodes long. However, these nodes are not selected keeping the destination in mind, and can often be geographically distant, causing any communication between them to pass through a large number of routers, over very inefficient paths. PHI and Dovetail, as non-overlay solutions, are inherently better than Tor in this regard. However, the use of helper nodes, which are not necessarily on the most efficient path between the source and destination, may result in excessively long paths. In comparison, TrafFu does not depend on overlays or intermediary nodes and allows the controller to employ any routing scheme as desired. Having a flexible routing policy typically leads to lower latency for the user and is a boost for **O3**.

8.2.3 Packet Headers

A major design consideration for the network privacy solutions (LAP, Hornet, PHI, Dovetail) was that they did not want to store packet state on the routers and stored them in the packet headers instead. This however, results in an increase in header length proportional to the length of the path and the size of the encryption key used. This is reflected in Table 8.1, which follows up on earlier analysis conducted in [11]. TrafFu on the other hand is similar to Tor in that we do not store packet state in the header. This improves the effective throughput, or goodput of our design and reduces the bandwidth overhead **O3**. However, unlike these designs, TrafFu comes with the heavy cost of increasing the flow table utilization on FEs. This increase is particularly significant when a larger number of keys are used.

Chapter 9

Conclusion

The paper introduces TrafFu, a design that leverages Software Defined Networks to enhance network privacy in an environment of untrusted forwarding elements. TrafFu encrypts the source and destination IP addresses in the packet header at the source, and relies on the controller to process and route encrypted packets such that the forwarding elements only ever see the ciphertext source-destinations. This helps us achieve Sender and Receiver anonymity within the network. Furthermore, TrafFu hinders powerful traffic analysis attacks by breaking the one-to-one mapping between plaintext and ciphertext source-destination addresses. While this results in an increase in flow table utilization, we show that at the cost of some privacy, we can reduce this effect by employing flow aggregation on encrypted addresses.

9.1 Future Work

TrafFu has been developed as a prototype and can be extended further. Primarily, to make our evaluation more comprehensive, we would like to implement TrafFu on real hardware, model real-world topologies and use real-world traffic traces. In doing so, we would gain a greater and more accurate understanding of the costs associated with our design. More specifically, in measuring the flow table utilization in a non-simulation setting, we would learn more about the deployability or feasibility of TrafFu. In the process, we would like to explore the possibility of finding a 'sweet-spot' for the tradeoff between privacy and flow-table overhead.

As discussed in Chapter 6, our claims of resilience against traffic analysis were theoretical. We would also like to deploy previously researched traffic analysis attacks on TrafFu and evaluate their effectiveness when conducted at different points in the network. In the event that our findings measure up to our theoretical expectations, we would also like to run the side-channel attacks mentioned earlier in Section 6.2, to fully explore an attackers options at deanonymizing TrafFu network traffic.

As referenced throughout the paper, we position TrafFu as a solution for privacy in the network layer. TrafFu does not concern itself with payload protection and leaves it to higher level protocols to do so. We note that this is but one deployment option and TrafFu does not necessarily have to be restricted as such. Subsequently, we would like to explore the option wherein payload obfuscation is provided as an option in TrafFu. This might require a greater co-ordination between the hosts and the controller or we could relax the threat model and implement it in a client-agnostic manner, relying on the first and last hop routers of the network. Moreover, if we would like expand the anonymity guarantees of

TrafFu to more than just mass surveillance and potentially incorporate targeted confirmation attacks, we could look to implement stronger defences such as leveraging the controller to analyze link utilization and deploy cover traffic accordingly to ensure a constant rate of traffic across the network.

Bibliography

[1] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and A. Selcuk Uluagac. Peek-a-boo: I see your smart home activities, even encrypted! *CoRR*, abs/1808.02741, 2018.

[2] Werner Almesberger et al. Linux network traffic controlimplementation overview, 1999.

[3] Randall Atkinson and Stephen Kent. Security architecture for the internet protocol. 1998.

[4] Richard Barnes, Bruce Schneier, Cullen Jennings, Ted Hardie, Brian Trammell, Christian Huitema, and Daniel Borkmann. Confidentiality in the face of pervasive surveillance: A threat model and problem statement. Technical report, 2015.

[5] George Dean Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. Privacy vulnerabilities in encrypted http streams. In *International Workshop on Privacy Enhancing Technologies*, pages 1–11. Springer, 2005.

[6] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 227–238, New York, NY, USA, 2014. ACM.

[7] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 605–616, New York, NY, USA, 2012. ACM.

[8] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, August 2007.

[9] Sambuddho Chakravarty, Angelos Stavrou, and Angelos D. Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *Proceedings of the 15th European Conference on Research in Computer Security*, ESORICS'10, pages 249–267, Berlin, Heidelberg, 2010. Springer-Verlag.

[10] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[11] Chen Chen, Daniele E Asoni, David Barrera, George Danezis, and Adrain Perrig. Hornet: High-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1441–1454. ACM, 2015.

[12] Chen Chen and Adrian Perrig. Phi: Path-hidden lightweight anonymity protocol at network layer. *Proceedings on Privacy Enhancing Technologies*, 2017(1):100–117, 2017.

[13] Heyning Cheng and Ron Avnur. Traffic analysis of ssl encrypted web browsing. *URL citeseer. ist. psu. edu/656522. html*, 1998.

[14] Mauro Conti, Qian Qian Li, Alberto Maragno, and Riccardo Spolaor. The dark side (-channel) of mobile devices: A survey on network traffic analysis. *IEEE Communications Surveys & Tutorials*, 20(4):2658–2713, 2018.

[15] Mauro Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security*, 11(1):114–125, 2015.

[16] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP '03, pages 2–, Washington, DC, USA, 2003. IEEE Computer Society.

[17] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.2. Technical report, 2008.

[18] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[19] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 332–346, Washington, DC, USA, 2012. IEEE Computer Society.

[20] Stephen Farrell and Hannes Tschofenig. Pervasive monitoring is an attack. 2014.

[21] M Fontanini. Libtins: packet crafting and sniffing library, 2016.

[22] Alan Freier, Philip Karlton, and Paul Kocher. The secure sockets layer (ssl) protocol version 3.0. Technical report, 2011.

[23] Yossi Gilad and Amir Herzberg. Spying in the dark: Tcp and tor traffic analysis. In *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*, PETS'12, pages 100–119, Berlin, Heidelberg, 2012. Springer-Verlag.

[24] Xun Gong, Negar Kiyavash, and Nikita Borisov. Fingerprinting websites using remote traffic analysis. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 684–686, New York, NY, USA, 2010. ACM.

[25] Andy Greenberg. The white house warns on russian router hacking, but muddles the message, 2018. `https://www.wired.com/story/white-house-warns-russian-router-hacking-muddles-message/`, Last accessed: Sept 2019.

[26] Jamie Hayes and George Danezis. K-fingerprinting: A robust scalable website fingerprinting technique. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 1187–1203, Berkeley, CA, USA, 2016. USENIX Association.

[27] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 31–42, New York, NY, USA, 2009. ACM.

[28] Hsu-Chun Hsiao, Tiffany Hyun-Jin Kim, Adrian Perrig, Akira Yamada, Samuel C. Nelson, Marco Gruteser, and Wei Meng. Lap: Lightweight anonymity and privacy. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 506–520, Washington, DC, USA, 2012. IEEE Computer Society.

[29] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: Transparent moving target defense using software defined networking. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 127–132, New York, NY, USA, 2012. ACM.

[30] Suman Jana and Vitaly Shmatikov. Memento: Learning secrets from process footprints. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 143–157, Washington, DC, USA, 2012. IEEE Computer Society.

[31] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 263–274, New York, NY, USA, 2014. ACM.

[32] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Díaz, and Matthew Wright. WTF-PAD: toward an efficient website fingerprinting defense for tor. *CoRR*, abs/1512.00524, 2015.

[33] Jeremy Kirk. Mikrotik routers targeted in data eavesdropping scheme, 2018. `https://www.bankinfosecurity.com/mikrotik-routers-targeted-in-data-vacuuming-scheme-a-11469`, Last accessed: Sept 2019.

[34] Hugo Krawczyk and Pasi Eronen. Hmac-based extract-and-expand key derivation function (hkdf). 2010.

[35] Diego Kreutz, Fernando MV Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[36] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[37] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 255–263, New York, NY, USA, 2006. ACM.

[38] Dwayne C Litzenberger. Pycrypto-the python cryptography toolkit. *URL: https://www. dlitz. net/software/pycrypto*, 2016.

[39] Liming Lu, Ee-Chien Chang, and Mun Choon Chan. Website fingerprinting and identification using ordered feature sequences. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security – ESORICS 2010*, pages 199–214, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[40] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[41] Deepankar Medhi and Karthikeyan Ramasamy. Network routing. *Algorithms, Protocols, and Architectures*, pages 60–82, 2007.

[42] M. Mendonca, S. Seetharaman, and K. Obraczka. A flexible in-network ip anonymization service. In *2012 IEEE International Conference on Communications (ICC)*, pages 6651–6656, June 2012.

[43] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 215–226, New York, NY, USA, 2011. ACM.

[44] Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, Andrew R. Curtis, and Sujata Banerjee. Devoflow: Cost-effective flow management for high performance enterprise networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 1:1–1:6, New York, NY, USA, 2010. ACM.

[45] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster protocol – version 2. *Draft, July*, 154:28, 2003.

[46] Rishab Nithyanand, Xiang Cai, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 131–134, New York, NY, USA, 2014. ACM.

[47] NOX Repo. The pox network software platform. *URL: https://github.com/noxrepo/pox*, 2017.

[48] ONF. Software-defined networking: The new norm for networks. Technical report, Open Networking Foundation, April 2012.

[49] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[50] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, pages 103–114, New York, NY, USA, 2011. ACM.

[51] Mike Perry. A critique of website traffic fingerprinting attacks. *Avaliable at: https://blog. torproject. org*, 2013.

[52] Andreas Pfitzmann and Marit Köhntopp. Anonymity, unobservability, and pseudeonymity — a proposal for terminology. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 1–9, Berlin, Heidelberg, 2001. Springer-Verlag.

[53] Barath Raghavan, Tadayoshi Kohno, Alex C. Snoeren, and David Wetherall. Enlisting isps to improve online privacy: Ip address mixing by default. In *Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*, PETS '09, pages 143–163, Berlin, Heidelberg, 2009. Springer-Verlag.

[54] Jody Sankey and Matthew Wright. Dovetail: Stronger anonymity in next-generation internet routing. In Emiliano De Cristofaro and Steven J. Murdoch, editors, *Privacy Enhancing Technologies*, pages 283–303, Cham, 2014. Springer International Publishing.

[55] Jeffrey Schiller. Strong security requirements for internet engineering task force standard protocols. Technical report, 2002.

[56] Tara Seals. Thousands of mikrotik routers hijacked for eavesdropping, 2018. `https://threatpost.com/thousands-of-mikrotik-routers-hijacked-for-eavesdropping/137165/`, Last accessed: Sept 2019.

[57] Andreas Sfakianakis, Christos Douligeris, Louis Marinos (ENISA), Marco Loureno (ENISA), and Omid Raghimi. ENISA threat landscape report 2018. Technical report, European Union Agency for Network and Information Security (ENISA), 2019.

[58] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *Proceedings of the 11th European Conference on Research in Computer Security*, ESORICS'06, pages 18–33, Berlin, Heidelberg, 2006. Springer-Verlag.

[59] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1928–1943, New York, NY, USA, 2018. ACM.

[60] Ahmed Sonba and Hassan Abdalkreim. Performance comparison of the state of the art openflow controllers, 2014.

[61] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 19–, Washington, DC, USA, 2002. IEEE Computer Society.

[62] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. Raptor: Routing attacks on privacy in tor. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 271–286, Berkeley, CA, USA, 2015. USENIX Association.

[63] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 44–, Washington, DC, USA, 1997. IEEE Computer Society.

[64] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. 01 2009.

[65] Matthias Wachs, Quirin Scheitle, and Georg Carle. Push away your privacy: Precise user tracking based on tls client certificate authentication. In *Proceedings of the Applied Networking Research Workshop*, ANRW '18, pages 3–3, New York, NY, USA, 2018. ACM.

[66] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 143–157, Berkeley, CA, USA, 2014. USENIX Association.

[67] Tao Wang and Ian Goldberg. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, pages 201–212, New York, NY, USA, 2013. ACM.

[68] Tao Wang and Ian Goldberg. On realistically attacking tor with website fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2016(4):21–36, 2016.

[69] Tao Wang and Ian Goldberg. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, pages 1375–1390, Berkeley, CA, USA, 2017. USENIX Association.

[70] Nicholas Weaver. Op-ed: Why the entire premise of tor-enabled routers is ridiculous, 2015.

[71] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 3–18, Washington, DC, USA, 2011. IEEE Computer Society.

[72] Matthew Wright, Micah Adler, Brian N. Levine, and Clay Shields. An analysis of the degradation of anonymous protocols, 2001.

[73] Dawn Xiaodong, Song David, and Wagner Tian. Timing analysis of keystrokes and timing attacks on ssh. 02 2002.

[74] Soheil Hassas Yeganeh. Simple distributed programming for scalable software-defined networks, 2015.

[75] Sebastian Zander and Steven J. Murdoch. An improved clock-skew measurement technique for revealing hidden services. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 211–225, Berkeley, CA, USA, 2008. USENIX Association.

[76] Kim Zetter. NSA laughs at PCs, prefers hacking routers and switches, 2013. `https://www.wired.com/2013/09/nsa-router-hacking/`, Last accessed: Sept 2019.

[77] Liancheng Zhang, Zhenxing Wang, Jiabao Fang, and Yi Guo. A sdn proactive defense scheme based on ip and mac address mutation. In Mengxing Huang, Yan Zhang, Weipeng Jing, and Amjad Mehmood, editors, *Wireless Internet*, pages 51–60, Cham, 2018. Springer International Publishing.