Evaluation of a Proposed Traffic-Splitting Defence for Tor

Using Directional Time and Simulation Against TrafficSliver

Utvärdering av ett Flervägsförsvar för Tor

Med Riktad Tid och Simulering mot TrafficSliver

Jonathan Magnusson <jonathan.majson@gmail.com>

Faculty of Health, Science and Technology

Master thesis in Computer Science

Second Cycle, 30 hp (ECTS)

Supervisor: Tobias Pulls, University of Karlstad, Karlstad, SWE <tobias.pulls@kau.se>

Examiner: Anna Brunström, University of Karlstad, Karlstad, SWE <anna.brunstrom@kau.se>

Karlstad, June 15th, 2021

Abstract

Tor is a Privacy-Enhancing Technology based on onion routing which lets its users browse the web anonymously. Even though the traffic is encrypted in multiple layers, traffic analysis can still be used to gather information from meta-data such as time, size, and direction of the traffic. A Website Fingerprinting (WF) attack is characterized by monitoring traffic locally to the user in order to predict the destination website based on the observed patterns. TrafficSliver is a proposed defence against WF attacks which splits the traffic on multiple paths in the Tor network. This way, a local attacker is assumed to only be able to observe a subset of all the user's total traffic. The initial evaluation of TrafficSliver against Deep Fingerprinting (DF), the state-of-the-art WF attack, showed promising results for the defence, reducing the accuracy of DF from over 98% down to less than 7% without adding artificial delays or dummy traffic. In this thesis, we further evaluate TrafficSliver against DF beyond what was done in the original work by De la Cadena *et al.* by using a richer data representation and finding out whether it is possible to utilize simulated training data to improve the accuracy of the attack. By introducing directional time as a richer data representation and increasing the size of the training dataset using a simulator, the accuracy of DF was improved against TrafficSliver on three different datasets. Against the original dataset provided by the authors of TrafficSliver, the accuracy was initially 7.1% and then improved to 49.9%. The results were confirmed by using two additional datasets with TrafficSliver, where the accuracy was improved from 5.4% to 44.9% and from 9.8% to 37.7%.

Keywords

Tor, Website Fingerprinting, Deep Learning, Deep Fingerprinting, Traffic Splitting, TrafficSliver

Sammanfattning

Tor är ett personlig-integritetsverktyg baserat på onion routing som låter sina användare anonymnt besöka hemsidor på internet. Även om trafiken är enkrypterad i flera lager, kan trafikanalys användas för att utvinna information från metadata som exempelvis: tid, storlek och riktning av trafik. En Website Fingerprinting (WF)-attack karaktäriseras av att övervaka trafik nära användaren för att sedan avgöra vilken hemsida som besökts utifrån mönster. TrafficSliver är ett föreslaget försvar mot WF-attacker genom att dela upp trafiken på flera vägar genom nätverket. Detta gör att en attackerare antas endast kunna se en delmängd av användarens totala trafik. Den första utvärderingen av TrafficSliver mot Deep Fingerprinting (DF), spjutspetsen inom WF-attacker, visade lovande resultat för försvaret genom att reducera träffsäkerheten av DF från över 98% till mindre än 7% utan att lägga till artificiella fördröjningar eller falsk trafik. I denna uppsats strävar vi att fortsätta utvärderingen av TrafficSliver mot DF utöver vad som redan har gjorts av De la Cadena *et al.* med en rikare datarepresentation och en undersökning huruvida det går att använda simulerad data för att träna attacker mot försvaret. Genom att introducera riktad tid och öka mängden data för att träna attacken, ökades träffsäkerheten av DF mot TrafficSliver på tre distinkta dataset. Mot det dataset som samlades in av TrafficSliver var träffsäkerheten inledelsevis 7.1% och sedan förbättrad med hjälp av riktad tid och större mängder av simulerad träningsdata till 49.9%. Dessa resultat bekräftades även för två ytterligare dataset med TrafficSliver, där träffsäkerheten blev förbättrad från 5.4% till 44.9% och från 9.8% till 37.7%.

Nyckelord

Tor, Website Fingerprinting, Deep Learning, Deep Fingerprinting, Traffic Splitting, TrafficSliver

Acknowledgements

First and foremost, I would like to thank my supervisor Tobias Pulls for his expertise in Tor and helpful feedback when writing this thesis. Secondly I want to thank Matthias Beckerle for sharing his knowledge about deep learning. Finally I would like to thank my friends and family for supporting me through my studies so far, resulting in this work.

Thank you.

Acronyms

BWR	*Batched Weighted Random*
CNN	*Convolutional Neural Network*
DF	*Deep Fingerprinting*
DL	*Deep Learning*
DT	*Directional Time*
GPA	*Global Passive Adversary*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IETF	*Internet Engineering Task Force*
ML	*Machine Learning*
MPTCP	*Multipath Transmission Control Protocol*
PET	*Privacy-Enhancing Technology*
RTT	*Round Trip Time*
TB	*Tor Browser*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
WF	*Website Fingerprinting*

Contents

Chapter 1

Introduction

In order to be anonymous on the Internet, for instance to protect your privacy or circumvent censorship, one can use *Privacy-Enhancing Technologies* (PETs) [7]. One choice is using a PET based on *Onion Routing*; i.e., connecting across multiple nodes through an encrypted circuit in order to unlink the sender from the receiver [8]. Even though the traffic is heavily encrypted in layers, a passive observer would still be able to monitor metadata such as time, size, and direction of the traffic. *Website Fingerprinting* [11] is the method of analysing the metadata of encrypted traffic near the user and differentiate between destination websites and this is within the threat model of Tor [5].

Tor is an onion routing PET which lets its users browse the web anonymously by connecting circuits through multiple nodes before reaching the destination. Each jump from one node to the next is encrypted in layers to hide the source and destination when the traffic is forwarded through the circuits. Tor is *not* designed to protect its users against a global attacker able to monitor all traffic going in an out of the network, but should at least unlink the source and the destination at a local level. If a local attacker is monitoring the traffic going *in* to the network they should not be able to figure out what the destination is. On the other hand, if the local attacker (close to the destination) is monitoring traffic coming *out* of the network they should not be able to see what the source is. The objective of Website Fingerprinting is to monitor the incoming traffic at a point local to the user (e.g., a WiFi hotspot, an internet service provider or the entry node in the Tor network itself) and use traffic analysis to predict what the destination is. The ability to find patterns in the traffic has evolved over the years, especially with the recent use of *deep learning* [15] to improve the attacks.

A repository of proposals for making changes to the Tor protocol was created to make the process more light-weight and structured[1]. In *Tor Proposal 329*, traffic splitting (i.e., splitting up and sending the traffic over multiple routes to its destination) is discussed for overcoming bottlenecks and improving performance[2]. The proposal

[1]`https://blog.torproject.org/tor-design-proposals-how-we-make-changes-our-protocol`, accessed 2021-05-24.

[2]`https://gitweb.torproject.org/torspec.git/tree/proposals/329-traffic-splitting.txt`,

brings up *Conflux* and *TrafficSliver* as methods of splitting the traffic in the Tor network. Conflux by AlSabah *et al.* [1] aims to improve the performance for users connecting to the Tor network via *bridges* (unadvertised nodes for circumventing censorship blacklists) by splitting the traffic on multiple circuits in the network. TrafficSliver by De la Cadena *et al.* [3] aims to protect the users of Tor from Website Fingerprinting attacks by splitting up the traffic on different paths, assuming that the attacker will only be able to monitor a subset of the total traffic coming from the user. The main difference of these traffic splitting approaches is the splitting strategies. Conflux dynamically measures paths in the network and split the traffic based on load balancing and congestion awareness. Contrarily, network performance was not the focus for the strategies of TrafficSliver. According to the results from TrafficSliver, the best performing splitting strategy against Website Fingerprinting attacks was to randomize the size of batches and select paths based on weights updating constantly as the traffic is sent.

The goal of this study is to further evaluate TrafficSliver as a Website Fingerprinting defence, beyond what was done in the original work by De la Cadena *et al.* [3]. TrafficSliver was analysing different splitting strategies against multiple deep learning-based Website Fingerprinting attacks, including the state-of-the-art: *Deep Fingerprinting* by Sirinam *et al.* [24], reducing its accuracy from more than 98% to less than 7% without adding dummy traffic or artificial delays.

1.1 Ethics and Sustainability

The *Tor Research Safety Board* has set up guidelines for minimizing privacy risks when studying the Tor network[3]. Firstly, this study does not run anything on the live Tor network, but utilize datasets from other peer reviewed publications [3, 24, 25] which falls under "auxiliary data" and "minimization". Secondly, these datasets were already publicly available (therefore not private) and did not contain any personal identifiable information or privacy sensitive data. In summary, we do not collect any new data and are not attacking real users, only reusing existing datasets and simulations.

Our ultimate goal of evaluating a Website Fingerprinting defence is to support the development of a safer Tor network, the results of this study are therefore openly published. By running attacks on proposed defences, stronger solutions can be considered and developed. A stronger defence will in turn prompt novel or stronger attacks and thus continue the cat and mouse game of security.

accessed 2021-05-24.

[3]`https://research.torproject.org/safetyboard/`, accessed 2021-05-24.

1.2 Method

De la Cadena *et al.* published a splitting-strategy simulator and a dataset from a TrafficSliver implementation in the live Tor network. By inspecting the dataset, the first step of this study is to find improvements in the form of a richer data representation which could be used to re-evaluate Deep Fingerprinting against TrafficSliver. The second step is to find out whether it is possible to utilize the splitting-strategy simulator to generate training data which could be used to train Deep Fingerprinting and test against data from the live Tor implementation collection. The third step is to modify the simulator to generate more training data from one single original dataset since the performance of deep learning models depend on both the quality and the size of the training dataset.

1.3 Delimitations

Firstly, while there are multiple Website Fingerprinting attacks (even multiple deep learning-based attacks [19, 23]) this study focuses on Deep Fingerprinting only, which has been considered the state-of-the-art attack in recent years. Secondly, both Conflux and TrafficSliver were mentioned in Tor Proposal 329, but the scope of this study is to only evaluate the latter and then discuss the implications this has on Conflux which is not built with Website Fingerprinting in mind.

This study utilize datasets from previous works in the field of Website Fingerprinting. One alternative approach would be to gather a completely new dataset. We also focus on changing the data representation instead of changing the attack model *per se* to improve the performance.

Website Fingerprinting has two main settings: *closed world* and *open world*, where closed world is easier to evaluate but not as realistic as the open world setting which has a lot more complex factors to take into consideration [13]. Since the dataset obtained from TrafficSliver only contained closed world traces, it was decided to leave open world Website Fingerprinting outside of the scope of this study.

1.4 Contributions

The contributions of this study takes the form of:

- An alternative version of TrafficSliver's splitting-strategy simulator where original timestamps are not necessary to generate split sub-traces. Our results shows similar improvements for the attacker when using the alternative version without original timestamps. This means that more datasets—those not containing timestamps—can be used for simulation.
- An improved evaluation of Deep Fingerprinting against TrafficSliver where:

- A richer data representation improves the accuracy on the dataset from TrafficSliver from 7.1% to 23.1%.
- We show that simulated datasets can be used to train the attack against non-simulated test data.
- Increased amounts of simulated training data improves the accuracy from 23.1% to 49.9% on the original dataset from TrafficSliver collected on the live Tor network. This indicates that there was previously not enough data for Deep Fingerprinting to improve against TrafficSliver.
- Simulated data from two additional common datasets in the area of Website Fingerprinting (Wang *et al.* [25] and Sirinam *et al.* [24]) also demonstrate an improvement in accuracy of the attack. Using richer data representation and increased amounts of simulated training data increase the accuracy from 5.4% to 44.9% for the first, and from 9.8% to 37.7% for the second. For the latter dataset, no original timestamps were available.

1.5 Outline

The reminder of the thesis is structured as follows. Chapter 2 describes the relevant background of Tor, Website Fingerprinting, deep learning and traffic splitting. In Chapter 3 the method of estimating conceptual gaps in traces with richer data representation is introduced, then the method of increasing the amount of training data using simulation is presented along with the datasets used in this study. The design and implementation of the TrafficSliver simulator and Deep Fingerprinting is presented in depth in Chapter 4. The results are shown and evaluated in Chapter 5. These results are then further discussed in Chapter 6 before concluding the thesis and going over future work in Chapter 7.

Chapter 2

Background

This chapter aims to provide the background necessary for understanding the method, concept, design and results of the study. First Tor is introduced with its architecture and threat model. Then we go over the concept of Website Fingerprinting and how it is used to fingerprint websites in Tor. Machine learning and deep learning are then touched upon before introducing Deep Fingerprinting and the notion of Directional Time. Then Traffic Splitting and two proposed methods of doing so in Tor are summarised, one of the methods being TrafficSliver; the focus of this study.

2.1 Tor

Tor is a low-latency overlay network which allows its users to anonymously establish *Transmission Control Protocol* (TCP) connections, called streams, over the Internet [5]. The Tor-network consists of *relays* (also known as *routers* or *nodes*) run by volunteers, across which a user can traverse through virtual *circuits* (i.e., multiplexed TCP streams). The relays, which forward traffic in the network, are further categorized into three different types depending on their location in a circuit: *guard relay*, *middle relay* and *exit relay*. The user first connects to a guard relay, which in turn extends the circuit to the middle relay, and the middle relay then extends the circuit to the exit relay which ultimately connects to the destination (see Figure 2.1.1). The traffic in Tor is sent in *cells* of 514 bytes[1] which are encrypted in multiple layers between the relays so that only the guard knows the IP-address of the user but not the destination, and only the exit knows the IP-address of the destination but not the user. At the time of writing, there are close to 7000 active relays in the Tor network[2].

Anonymous communication comes with a trilemma (i.e., pick only two from three options): *(i) strong anonymity*, *(ii) low bandwidth overhead* and *(iii) low latency* [4]. Due to its low-latency and low-bandwidth overhead properties, enabling the users to

[1]`https://github.com/torproject/torspec/blob/master/tor-spec.txt#L68`, accessed 2021-05-24.

[2]`https://metrics.torproject.org/networksize.html`, accessed 2021-05-24.

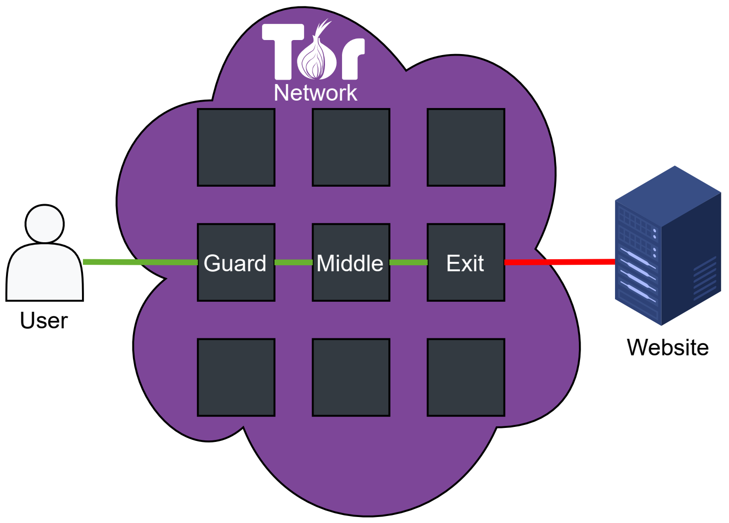

Figure 2.1.1: Example of fully established circuit in the Tor network. The green connections indicate the encryption provided by Tor, and the connection between the exit and the destination might use HTTP or HTTPS.

interactively browse on the web, the Tor network is *not* designed to provide strong anonymity, i.e., defend against a *Global Passive Adversary* (GPA), who is able to see all traffic on the Internet and correlate the timing of traffic going in and out of the *entire* Tor network.

Users typically use *Tor Browser* (TB)[3], a fork of Firefox that transparently routes the traffic through Tor. The *Hypertext Transfer Protocol Secure* (HTTPS) connection established between TB and the website stretches end-to-end within the Tor circuit, which creates a multi-layer of security. The cryptographic protocol used in HTTPS is *Transport Layer Security* (TLS)[4] which at the first layer encrypts the plaintext HTTP traffic sent from the client to the destination. Through the Tor network another layer of encryption is added to the cells sent from the client to the exit relay. As a third layer of encryption the relays maintain TLS connections to each other (built and persisted as used), in order to send cells as parts of streams in the circuits.

Additionally, Tor implements *Onion Services* which are typically reachable over circuits through six relays instead of three. These circuits do not contain any exit relay since the traffic does not “exit” the Tor network. The Onion Service protocol not only keeps the user anonymous from the service, but this is also the case in the opposite direction. Both the user and the Onion Service can send traffic over Tor without knowing the IP-address of each other. *Website Fingerprinting* attacks (see Section 2.2) is a larger threat against Onion Service websites compared to normal websites due to the relatively smaller number of services available. In this study the focus is on

[3]`https://www.torproject.org/download/`, accessed 2021-05-24.
[4]`https://tools.ietf.org/html/rfc8446`, accessed 2021-05-24.

"regular" circuits used to browse the "regular" web, but the results are applicable for Onion Services as well.

2.2 Website Fingerprinting

Even though the traffic in Tor is encrypted, analysis of metadata such as amount, direction and time of network traffic—known as traffic analysis—can be used to attack Tor in different ways. One such attack is *Website Fingerprinting* (WF) [11]. A *local adversary* is able to monitor the patterns of encrypted traffic of a user when visiting a specific destination. These patterns include timestamps, direction (outgoing or incoming traffic) and size, but since Tor sends the encrypted traffic in *equally-sized* cells, the size does not vary and is therefore ignored when looking for patterns in Tor traffic. As opposed to a GPA, a local passive adversary is close to the client, being an internet service provider, router, WiFi hotspot or a Tor guard relay for example. When a user sends traffic through the Tor network, and the adversary passively monitors the encrypted traffic between the guard and the user, or within the guard itself (see Figure 2.2.1), it would be possible to classify any traces if they have been observed previously. A WF attack could also be conducted at some point between the guard and the exit relay [12], but in that case the user's IP-address remains unknown (only the guard relay knows about it).

In WF there are two main settings: *closed world* and *open world*. The closed world setting contains a set of x websites. The adversary can visit the x websites and monitor the traffic. The goal of the adversary is then to monitor the traffic of a user at the guard (or between the guard and the user) and guess which of the x websites they were visiting. The key point here is that the user can only visit one of the x websites and no other. In contrast, the open world setting contains a set of y websites in addition to the previous x websites. Just as in closed world, the adversary can visit the x websites and monitor the traffic. But the user can then visit either a website from set x or from set y. This means that the goal of the adversary is first to figure out whether the user has been visiting a website from x or y, and if they have been visiting a website in set x, then the goal of the adversary is to figure out which one. In closed world there are x classes of websites, while in open world there are typically $x + 1$ classes (one additional class covering set y). In WF set x is referred to as *monitored* traces and set y as *unmonitored* traces. The open world setting is more realistic and therefore considers more complex challenges compared to the closed world setting [13]. The scope of this study is to evaluate the closed world setting of WF since the dataset from TrafficSliver (see Section 2.4.2) was set in the closed world.

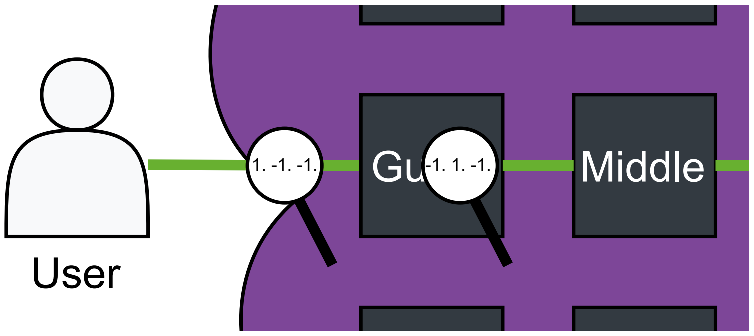

Figure 2.2.1: Website Fingerprinting by locally analysing patterns in the encrypted traffic at the guard or between the user and the guard (internet service provider, router, WiFi hotspot, etc.).

2.3 Machine Learning and Deep Learning

WF as a classification-based attack has been studied for a decade using classical *Machine Learning* (ML), with increasingly more complex feature engineering and improved results [9, 20, 21, 25]. ML is a subset of Artificial Intelligence and is best described by Tom M. Mitchell [18]:

> *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E".*

This means that the algorithm gains experience by training on a task and that this experience can be measured in performance. The more the algorithm trains and gains experience, the better it performs on the task. These tasks are usually classification tasks in ML (e.g., classify mail as spam or separating images of cats and dogs).

More recently, a subset of the ML field has gotten the attention of WF research, namely *Deep Learning* (DL). While ML needs to perform the feature extraction manually from the input before classification, DL automatically extracts the features from the underlying data [15]. Due to its feature extraction, the DL model can work on both structured and unstructured data as input, and this in turn has made DL efficient in object detection and speech recognition, both of which are classification problems (e.g., does the *sound* match any known *word*).

Convolutional Neural Network (CNN) is a class of DL used primarily for computer vision, which can be applied in a plethora of cases, including: medical image analysis [16], agriculture monitoring [17] and object detection in cars [2]. CNNs uses backpropagation algorithms to adjust the parameters of hidden layers (between the input and output layers) during training and the architecture is analogous to the pattern of connected neurons in the biological brain. A neuron receive input from other neurons and will only forward a signal when reaching a threshold. In addition to helping doctors with medical images and detecting cars on the road, what if a CNN could differentiate between network traces?

2.3.1 Deep Fingerprinting

Sirinam *et al.* [24] proposed in 2018 a DL design based on a specialized CNN for fingerprinting websites efficiently, even against strong state-of-the-art practical defence mechanisms proposed for Tor, such as the WTF-PAD defence by Juarez *et al.* [14]. Sirinam *et al.* defines a trace as the direction (outgoing or incoming) of up to 5000 equally-sized encrypted cells, which are represented by a positive or negative value of 1 respectively, shown in Figure 2.3.1. The proposed novel neural network for WF called *Deep Fingerprinting* (DF) takes the traces monitored between the user and the guard relay as input and presents the predicted website as output, as can be seen in Figure 2.3.2.

$$[1. \quad -1. \quad -1. \quad ... \quad -1. \quad -1. \quad -1.]$$

Figure 2.3.1: Example trace of outgoing and incoming cells. Each trace contains up to 5000 cells padded with zeroes.

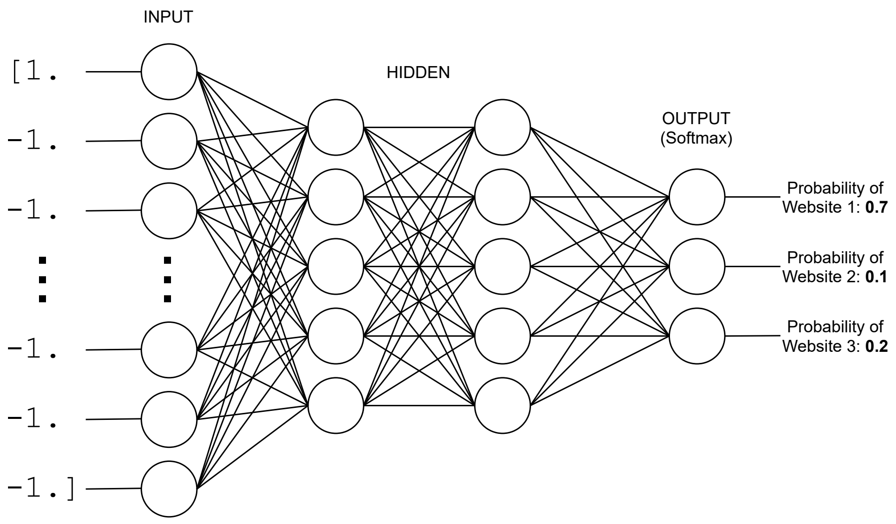
INPUT
HIDDEN
OUTPUT (Softmax)
[1.
-1.
-1.
-1.
-1.
-1.]
Probability of Website 1: **0.7**
Probability of Website 2: **0.1**
Probability of Website 3: **0.2**

Figure 2.3.2: The trace is fed into the input of the model. Any layers between the input and the output are called hidden layers. The output layer applies a softmax function to the result in order to calculate the probability of a given website.

2.3.2 Directional Time

Rahman *et al.* [22] presented a notion of combining the directions of the trace in DF and the timestamps available when monitoring, by multiplying the two values. This notion of *Directional Time* (DT) adds an additional dimension of input for attacks such as DF. Figure 2.3.3 shows an example where the vector a (direction) and vector b (time) are iterated in parallel and multiplied in vector c (directional time).

The closed world setting evaluation of DF using DT achieved a better accuracy compared to only using direction or only using time. The results show an accuracy

```
a) [1.   -1.   -1.   ...    -1.     -1.     -1.]
b) [1.    5.    8.   ...   103.    174.    212.]
c) [1.   -5.   -8.   ...  -103.   -174.   -212.]
```

Figure 2.3.3: Multiplying direction (a) and timestamps (b) to get Directional Time (c).

of 98.4% against undefended Tor traffic and 93.5% against the WTF-PAD defended traffic, which was several points better than the previous results using only direction (90%). The study concluded that using time as an additional input results in stronger WF attacks.

2.4 Traffic Splitting

The concept of traffic splitting is characterized by sending the traffic from the source to the destination via multiple paths. This could improve performance by circumventing network bottlenecks or from a security and privacy perspective assume that a local attacker is only able to monitor a subset of the total user traffic. An ongoing effort by the *Internet Engineering Task Force* (IETF)[5] is to integrate multipath in TCP, unsurprisingly called *Multipath Transmission Control Protocol* (MPTCP)[6]. The main focus of MPTCP is to improve performance, this includes network redundancy and a higher bandwidth.

Traffic splitting is currently under investigation in Tor Proposal 329[7]. One important aspect of traffic splitting are the points of splitting and merging the traffic. The split could occur at the point of the user, setting up multiple guards in the network or at the guard itself. The merge could be done at the middle or exit relay which would then continue send merged traffic to the exit or destination respectively. This investigation is influenced by both *Conflux* (see Section 2.4.1) and *TrafficSliver* (see Section 2.4.2) with different implications on performance and anonymity.

2.4.1 Conflux

AlSabah *et al.* [1] created a traffic splitting scheduler in 2013 with the focus of increasing performance in the Tor network. Its primary use case is to increase the throughput for users connecting to the network using *bridges* (i.e., unadvertised guard relays used for circumventing Tor censorship). These bridges are known to have generally low bandwidth which could be improved by splitting the traffic on different paths in the network. According to the proposal, the user would split the traffic on multiple guards which then merge at a common exit relay, shown in Figure 2.4.1.

[5] `https://www.ietf.org/`, accessed 2021-05-24.

[6] `https://tools.ietf.org/html/rfc8684`, accessed 2021-05-24.

[7] `https://gitweb.torproject.org/torspec.git/tree/proposals/329-traffic-splitting.txt`, accessed 2021-05-24.

The splitting strategy of Conflux, based on congestion awareness and load balancing, dynamically measures the different paths and then sends cells based on throughput. In their paper, AlSabah *et al.* also underline that there is a slight trade-off between performance gain and user anonymity. Their results indicate up to 75% improvement in total download time and a 30% improvement in the Tor client's queueing delay.

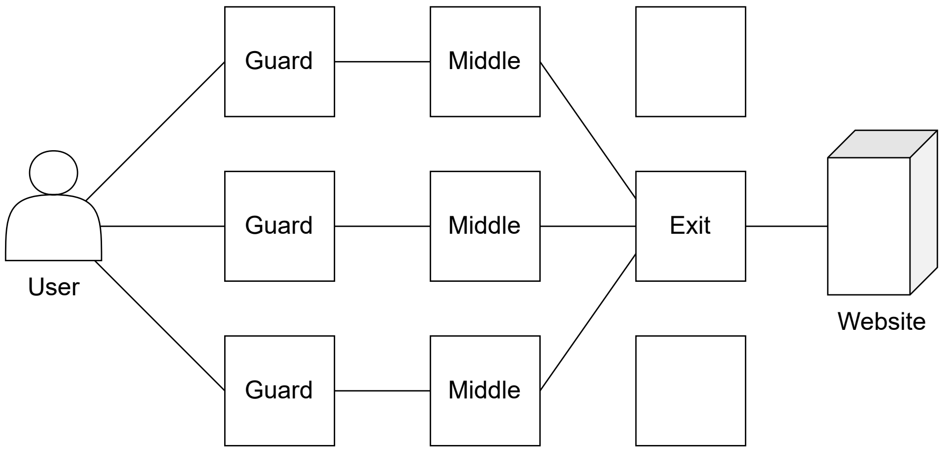

Figure 2.4.1: Conflux using three guards (bridges) to split traffic.

2.4.2 TrafficSliver

De la Cadena *et al.* [3] proposed a WF defence called TrafficSliver in two different versions: *TrafficSliver-App* on the Application layer and *TrafficSliver-Net* on the Network layer. The TrafficSliver-App method does not require any modifications to the Tor relays to deploy. It acts like an additional proxy between the browser and the onion proxy of the user and establishes multiple complete circuits (guard, middle and exit relays) to the destination. In order to split the traffic on these circuits, TrafficSliver-App utilizes the *range option* in HTTP/1.1 protocol for a partial request. The TrafficSliver-Net method introduces changes to the Tor protocol in order to be deployed. The user's onion proxy splits the traffic on multiple guards which merge together at a common middle relay (see Figure 2.4.2 for an example) with the help of new types of Tor cells and cookies, inspired by the rendezvous cookies used for Onion Services.

Instead of sending the Tor cells *Randomly* to the guards, a *Weighted Random* approach increases the diversity of the traffic distribution. For each page load, a vector $\vec{w}$ containing m probabilities for each point of splitting. These probabilities are computed from a Dirichlet distribution[8] of m dimensions [6], selecting which guard each cell is sent to. Additionally, in *Batched Weighted Random* (BWR), the cells are grouped in *batches* of size n. The value of n is then updated after each batch, constantly

[8]A distribution on probability distributions.

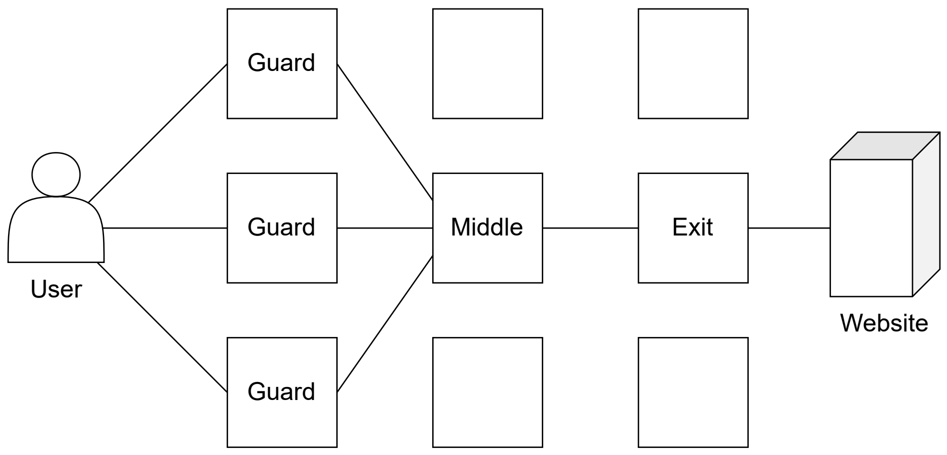

Figure 2.4.2: TrafficSliver-Net using three guards to split traffic.

during a single page load. These traffic splitting schedulers, including *round robin*, were evaluated, and the best performing splitting strategy against multiple WF attacks (DF being one of them) was BWR using 5 guards (BWR5). TrafficSliver, using BWR5, managed to reduce the accuracy of DF from above 98% to less than 7%. A couple of previously proposed defences against WF attacks had added either dummy traffic or artificial delays in order to defend the traffic [14, 26], but the BWR splitting strategy does not add any overhead.

The authors of TrafficSliver also created a splitting-strategy simulator available on GitHub[9]. This simulator takes non-split traces as input in the Wang-format [25], which is described as one folder for each label (website) and one file for each sample trace. Each trace-file have metadata of cells on each newline consisting of a direction, a timestamp and optionally a size. The simulator then splits the trace according to one of 11 schemes (variants of round robin, random, weighted random and batched weighed random). Every cell in each trace is allocated to a path based on the chosen scheme, then the simulator randomly picks a latency for each simulated path from a collection of measurements done by TrafficSliver. These latencies—measured as *Round Trip Time* (RTT)—are then used to replace the real time difference between a sent and a received cell. A small delta-time is also used to (i) offset cells in bursts in either direction and (ii) denote the time between receiving and then sending a cell at the user. After the time adjustments, the cells are grouped by their allocated path and sorted on their new timestamps. The output of the simulator is a set of sub-traces which amounts to the number of input traces times the number of paths used in the simulation. An input of 10,000 traces when using a splitting scheme with 5 paths would result in 50,000 sub-traces as the output.

[9]`https://github.com/TrafficSliver/splitting_simulator`, accessed 2021-05-24.

In addition, the authors of TrafficSliver shared a real world dataset of traces using the BWR5 splitting scheme. This data was collected by using a modified version of Tor, implementing the TrafficSliver-Net WF defence[10]. The dataset is organised as one file for each label and one trace on each newline. The cells were separated by whitespace and contained timestamp, direction, size and the IP-address of the chosen guard. The dataset contained traces from 100 websites and 100 traces from each website which amounts to a total of 10,000 traces.

[10]`https://github.com/TrafficSliver/trafficsliver-net`, accessed 2021-05-24.

Chapter 3

Method

In Tor Proposal 329 both TrafficSliver and Conflux were used for reference when discussing traffic splitting in Tor. Since De la Cadena *et al.* evaluated TrafficSliver using different strategies specifically against Website Fingerprinting (which was not the focus of Conflux), our decision was to evaluate TrafficSliver and then discuss the implications of the results for Conflux. Another benefit of picking TrafficSliver over Conflux is the access to a splitting-strategy simulator *and* a real world dataset from the former. We decided to only use previous datasets and not to collect new data from the Tor network, we can thus directly compare performance of non-defended datasets with their respective original studies. We also decided to focus on data representation and not to improve the attack models *per se*. This chapter first introduce the idea of conceptual sub-trace gaps, then the method of increasing the dataset size with simulation and the datasets used in this study are presented.

3.1 Sub-trace Gaps

From the authors of TrafficSliver we got access to a dataset with traces from the Tor network using their BWR5 splitting method. Each cell in the trace consist of a timestamp, a direction, a size and the IP-address of the guard it is going to. Ignoring the different guard IP-addresses of each cell leaves us with a non-split trace (presented in more detail in Section 3.2 as the dataset named *DS-TS*). Taking the guard IP-addresses into consideration gives us two sub-trace variants: (i) a sub-trace in which we see the cells from a given guard and do *not* include anything from the other guards, and (ii) a sub-trace in which we see the cells from a given guard and treat any other cells as unknown (*empty gaps*). An example with 3 guards is shown in Figure 3.1.1.

The conceptual gaps in the second variant of the sub-trace indicates that another guard is processing an unknown cell. This information is not something that a local passive observer can gather from watching one single path using *only direction* of each cell, but adding *time* could partially replace the missing information of the gaps.

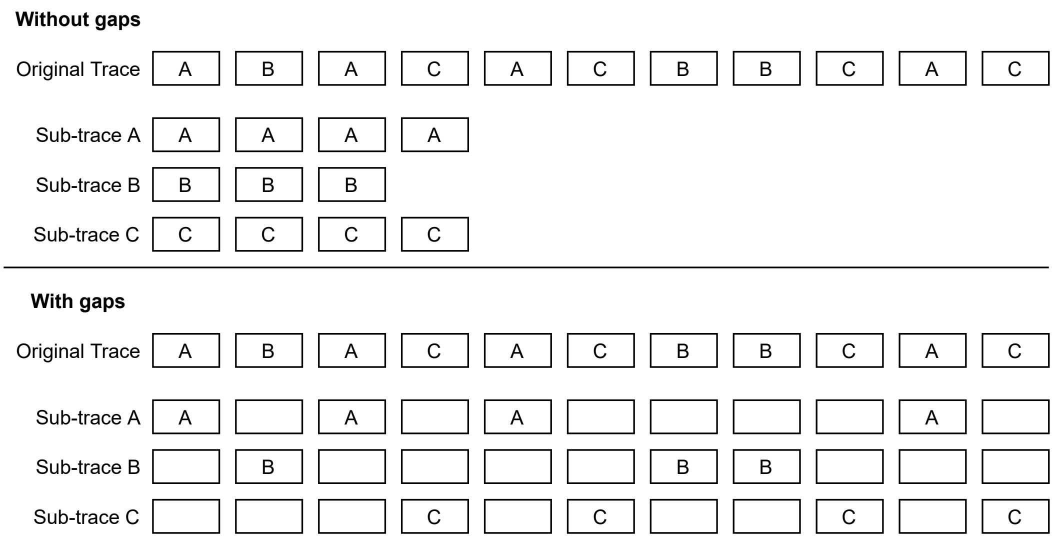

Figure 3.1.1: Sub-trace variants example with 3 guards (A, B, C).

Based on the findings by Rahman *et al.* [22], DL WF attacks such as DF could improve by adding time to the direction already used when representing traces (i.e., using DT). Without time, these conceptual gaps are not observable to a local passive observer monitoring a subset of the traffic. By using DT in the traces, an attacker could discern that a large time interval between cells might indicate that some cells are sent to other guards instead.

3.2 Simulation of Datasets

Due to the nature of splitting the trace on multiple paths, each sub-trace contains less data. This is especially true since the split is based on randomness, i.e., the trace is not evenly distributed on each sub-trace (BWR splitting strategy in Section 2.4.2). A trace of 100% could for example be split into five sub-traces of 96%, 1%, 1%, 1%, 1%. Since the attacker can monitor their own traffic when visiting a website from the monitored set, they can always obtain a non-split trace of a given website. The attacker could then use the available splitting-strategy simulator from TrafficSliver to generate sub-traces. In addition, the attacker can run the simulator multiple times with different seeds for the randomness and thus multiply the number of sub-traces available from one single original trace. This means that it is possible to generate large amounts of training data for DL WF attacks. When using the simulator from TrafficSliver to generate sub-traces, it is first of all important to test against a non-simulated dataset to make sure that the simulator generates valid training data. Then, after that, it is possible to introduce additional datasets for the simulator to see if the same method can be used on other types of datasets.

Three different datasets were used as input for the splitting-strategy simulator: *DS-TS*, *DS-DF* and *DS-WA*. Dataset DS-TS is the dataset from TrafficSliver[1] which is merged together into a non-split dataset by ignoring the IP-address of each guard. DS-WA is the dataset used by Wang *et al.* [25] when studying ML-based attacks in WF[2]. DS-DF is the dataset collected by Sirinam *et al.* [24] for evaluating DF[3]. Both open world and closed world datasets were available for DS-DF, but only the closed world dataset was used due to the scope of this study. A summary of each dataset is shown in Table 3.2.1.

Name	Labels	Traces	Cells	Avg Cells/Trace	Contains
DS-TS	100	10,000	40,326,982	4032.7	direction & time
DS-WA	100	9,000	16,268,626	1807.6	direction & time
DS-DF	95	95,000	174,917,787	1841.2	direction

Table 3.2.1: Closed world datasets used as input for the splitting-strategy simulator.

The DS-DF is the only dataset of the three not containing any original timestamps. This is one of the reasons why SIM-B was introduced (see Section 4.1 below), since it calculates relative timestamps based only on the latencies and delta-time provided in the simulator.

[1]https://www.informatik.tu-cottbus.de/~andriy/zwiebelfreunde/TrafficSliver-Net.tar.xz, accessed 2021-05-24.

[2]http://home.cse.ust.hk/~taow/wf/data/knndata.zip, accessed 2021-05-24.

[3]https://github.com/deep-fingerprinting/df#dataset, accessed 2021-05-24.

Chapter 4

Design and Implementation

This chapter first describes the design and implementation of the modified TrafficSliver splitting-strategy simulator. Then the details around the implementation of Directional Time are introduced. Finally the implementation of Deep Fingerprinting is presented along with the specification of the environment running both the simulator and Deep Fingerprinting.

4.1 TrafficSliver Simulator

Some of the modifications to the splitting-strategy simulator from TrafficSliver were:

1. For simplicity, everything but the BWR simulation was stripped away since it was the best performing splitting strategy against DF.
2. The simulator was fixed to handle an unexpected behavior with RTT when the trace started with a negative direction (-1).
3. The simulator was modified to run with the same seed for randomness each first time, and then use a nonce (counting from 0 and up) as additional seed to run the simulator in a loop, creating more simulated sub-traces.

Sirinam *et al.* structured their dataset in pkl-files[1], where one file would contain the traces and another would contain the labels. One of the design choices was therefore to include a way to read from pkl-files. The original splitting-strategy simulator stored the split traces in Wang-format, where each sub-trace was stored in files with the direction and time of every cell (separated by whitespace) on each line. In order to be space efficient, the simulator was modified to save the result after each iteration in separate directories in a Wang-like format (not separating direction and time but using DT). The DF implementation (see Section 4.3) then loads the dataset from these directories based on a size parameter. The traces are stored with gaps and DT from

[1]`https://docs.python.org/3/library/pickle.html`, accessed 2021-05-24.

which variants of the dataset (DT, gaps, DT+gaps and without time) can be extracted in the DF implementation.

After these changes the simulator was split into two variants. *SIM-A* which uses the original method of simulating the time between cells (Appendix A.1), and *SIM-B* which ignores the original timestamps of the cells and only uses the latency and delta-time provided in the simulator to calculate the relative time to the first cell, which starts at 1 instead of its original timestamp (Appendix A.2). The implications of not using the original timestamps in SIM-B is discussed in Chapter 6 based on the results in Chapter 5. Both versions of the simulator use the same splitting-strategy (BWR5) to split each input trace into five sub-traces.

The advantage of SIM-B is that datasets without original timestamps (only the direction of each cell) can be used with the simulator, since the first cell starts at 1 and the timestamp of every subsequent cell is dependent on the latency or the inter-cell delta-time. One such dataset without timestamps is the *DS-DF* dataset from Sirinam *et al.* which was introduced earlier in Section 3.2.

4.2 Directional Time

Each trace used for DF initially and TrafficSliver only used the direction of each cell (even though timestamps were available for the latter). Multiplying the direction with the timestamp results in DT, as described earlier in Section 2.3.2. In order to multiply the timestamp with the sign of the direction, the timestamp can not be zero, since zero has no sign. For sub-traces containing conceptual gaps, the timestamp is not normalized if a zero is found. The time is normalized to be relative to the first cell with a timestamp, so its new timestamp is therefore set to 1 (before multiplying with the direction) and every single subsequent cell timestamp is set relative to it. The implementation is shown in Listing 4.1.

```python
def normalize_trace(trace):
    new_trace = []
    zero_time = 0
    for x in range(len(trace)):
        if trace[x] == 0: # skip DT on zeroes
            new_trace.append(0)
        else:
            if zero_time == 0:
                zero_time = trace[x]
            direction = 1 if trace[x] > 0 else -1
            new_time = abs(abs(trace[x]) - abs(zero_time))
            new_trace.append((new_time + 1) * direction)
    return new_trace
```

Listing 4.1: Function for normalizing directional time in the simulator.

4.3 Deep Fingerprinting

A PyTorch[2] implementation of DF was used for training and testing with the datasets in this study[3]. This implementation was modified to load the output format from the simulator as well as extracting traces with gaps and/or directional time. It supported loading data via a specified extract function. It was therefore easy to implement a new dataset loader function (Listing 4.2) and a new extract function for the output created by the simulator.

```
def load_dataset_BWR5(dir_path, length, _dt, _gaps, ds_size, extract_func):
    data = {}
    labels = {}
    sample_tracker = [0] * 100

    for i in range(ds_size):
        print(f"loading set: {i+1}/{ds_size}")
        iter_path = os.path.join(dir_path, str(i))
        for website in tqdm(os.listdir(iter_path)):
            web_path = os.path.join(iter_path, website)
            for fname in os.listdir(web_path):
                if '.cell' in fname:
                    file_path = os.path.join(web_path, fname)
                    with open(file_path, "r") as f:
                        trace = extract_func(f.read(), length, _dt, _gaps)
                        label, sample = fname.split('.')[0].split('-')
                        ID = f"m-{label}-0-{sample_tracker[int(label)]}"
                        data[ID] = trace
                        labels[ID] = int(label)
                        sample_tracker[int(label)] += 1

    return data, labels, sample_tracker
```

Listing 4.2: New dataset loader function for BWR5 datasets.

The extract function for the BWR5 traces can be seen in Listing 4.3. In addition, a new splitting function was also implemented (Appendix B.1), splitting on samples instead of the original partitions and without handling an unmonitored dataset since the scope of the study is closed world. The dataset is split into 80% training, 10% validation and 10% testing. The training and validation data is used to fit the model in epochs. To avoid overfitting on the validation set at the end of each epoch, a testing set—never used during training—is used to test if the model can generalise the patterns found when training.

[2] https://pytorch.org/, accessed 2021-05-24.
[3] https://github.com/pylls/padding-machines-for-tor/blob/master/evaluation/once.py, accessed 2021-05-24.

```python
def extract_BWR5(trace, length, dt, gaps):
    data = np.zeros((1, length), dtype=np.float32)
    trace = trace.split("\n")
    i = 0
    for n in range(len(trace)-1):
        if gaps and dt:
            data[0][i] = np.float(int(trace[n]))
            i += 1
        elif gaps and int(trace[n]) == 0:
            data[0][i] = 0.0
            i += 1
        elif dt and int(trace[n]) != 0:
            data[0][i] = np.float(int(trace[n]))
            i += 1
        elif int(trace[n]) > 0:
            data[0][i] = 1.0
            i += 1
        elif int(trace[n]) < 0:
            data[0][i] = -1.0
            i += 1
        # else, found a zero and gaps is false
        if i >= length:
            break

    return data
```

Listing 4.3: Extract function for the BWR5 datasets.

Both the splitting-strategy simulator and the DF implementation were running on the following system:

- **Operating system:**
  Linux kernel 5.4.0-65-generic #73-Ubuntu SMP x86_64 GNU/Linux
- **GPU:** GeForce RTX 2080 Ti
- **CPU:** Intel(R) Core(TM) i9-9900X CPU @ 3.50GHz
- **Memory:** 64 GB RAM

All code is available on GitHub[4].

[4]https://github.com/Arcnilya/trafficsliver-evaluation

Chapter 5

Evaluation and Result

First the datasets from Section 3.2 were used with the DF implementation to compare the accuracy of previous corresponding studies where the datasets originate from. The accuracy for each non-defended dataset against DF is shown in Table 5.0.1 using only direction (as the previous studies have done). The accuracy of DF-TS (96.8%) is within two percentage points of the closed world non-defended result by De la Cadena *et al.* (98.75%). The accuracy of DS-WA (92.2%) does not have a previous result using the DF attack to compare with. The accuracy of DS-DF (98.2%) corresponds well with the closed world non-defended result by Sirinam *et al.* (98.3%). DF is considered performing well when the accuracy is above 90%. The slight differences between the results could be a result from splitting the data differently into training and testing.

Dataset	DS-TS	DS-WA	DS-DF
Accuracy	96.8%	92.2%	98.2%

Table 5.0.1: Accuracy for non-defended datasets (using only direction, not time).

Figure 5.0.1 shows how the accuracy of DF against TrafficSliver, using the DS-TS dataset with SIM-A, is affected by introducing DT and increasing the amount of sub-traces. A baseline of only using direction and not time is shown as the bottom line in the graph (**without time**), which at 50,000 sub-traces (10,000 traces split in five) results in an accuracy of 7.1%. Increasing the amount of sub-traces with the help of the simulator increases the accuracy to 12.4%. The top two lines in the graph represents the use of conceptual gaps in the sub-traces when using the BWR5 splitting-strategy, with DT (**DT gaps**) and without (**gaps**). This is conceptually an upper limit of accuracy as a reference when estimating the gaps with DT. The accuracy of using conceptual gaps at 50,000 sub-traces is 50.3% for **gaps** and 58.8% for **DT gaps**. When increasing the amount to 600,000 sub-traces, the accuracy is increased to 63.4% and 74.6% respectively. The remaining lines in the graph represent the simulated (**DT sim**) and real (**DT real**) accuracy for DT. These are separated since the testing is also done with the *non-simulated* dataset available from TrafficSliver. By comparing the accuracy on simulated data and the accuracy on non-simulated data we can see if the simulator is a good representation of the real world TrafficSliver implementation. Looking at the accuracy at 50,000 sub-traces shows that the simulation accuracy and the real accuracy are increased to 24.8% and 23.1% respectively. The testing accuracy of DF with DT increase to 53.2% for **DT sim** and 49.9% for **DT real** when using 600,000 sub-traces (12 times the original size).

The increase of accuracy when introducing DT indicates that DF benefits from the richer data representation. The trend of using DT with increasing amount of training data is relatively similar for testing on both simulated data and non-simulated data, showing that DF is able to learn how the defence works by training on simulated data. The slight difference in accuracy between only using conceptual gaps in the sub-traces and using conceptual gaps *and* DT indicates that DF can extract useful time information from cells between gaps as well.

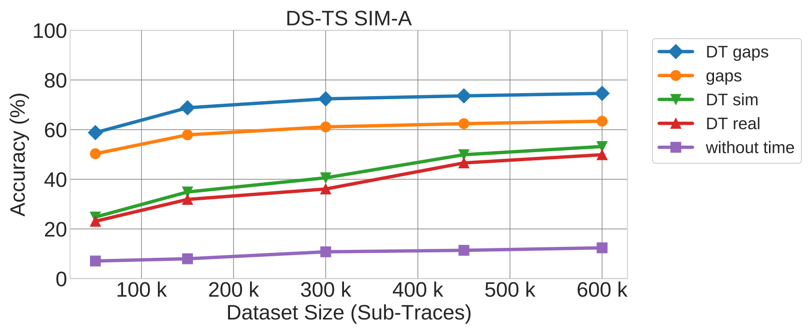

Figure 5.0.1: SIM-A using DS-TS as input. The testing of DT is also done against the non-simulated BWR5 dataset.

Figure 5.0.2 shows the accuracy of DF against TrafficSliver using the DS-WA dataset with the SIM-A version of the simulator. Similarly to DS-TS, the accuracy of DF when only using direction (**without time**) goes from 5% to 11% when increasing the amount of sub-traces. The upper limit of conceptual gaps at 45,000 sub-traces (9,000 traces split in five) yields an accuracy of 45.6% for **gaps** and 48.3% for **DT gaps**. This accuracy is gradually improved to 57.8% and 64.2% respectively when increasing the amount of sub-traces. In contrast to DS-TF in Figure 5.0.1 there is only one line for DT in DS-WA (**DT**). This is because there are no real world sub-traces of DS-WA for testing. At 45,000 sub-traces the accuracy is increased from 5.4% to 14.1% by introducing DT. The accuracy is then increased further to 44.9% when also increasing the number of sub-traces to 12 times the original amount.

Here similarities with DS-TS can be found. Using a richer data representation is increasing the accuracy of DF. The slight difference in accuracy of using conceptual gaps with and without DT indicates that DF can extract additional information from bursts of cells.

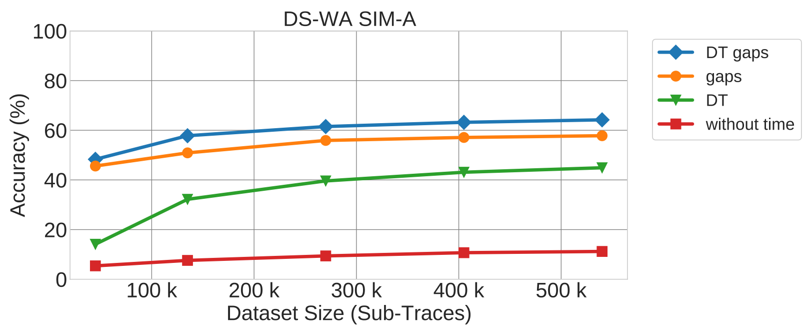

Figure 5.0.2: SIM-A using DS-WA as input.

Figure 5.0.3 shows the accuracy of DF against TrafficSliver using the DS-WA dataset with the SIM-B version of the simulator. The DF accuracy when only using the direction of the cells in the trace (**without time**) goes from 5% to 12% when increasing the sub-trace amount. The accuracy of conceptual gaps (**gaps**) goes from 53.1% to 64.5% when increasing the amount of sub-traces. When introducing DT the accuracy (**DT**) jumps from 5.6% to 11.2% at 45,000 sub-traces which is further increased to 35.5% using 12 times the original amount of sub-traces. The most significant disparity between DS-WA SIM-A and DS-WA SIM-B is the size of the distance between the accuracy of DT and the accuracy of conceptual gaps. There is no line for DT *and* gaps combined, because no original timestamps are used in SIM-B (discussed in more detail in Section 6.1).

When comparing SIM-A and SIM-B on DS-WA, the main difference is the accuracy of DF when utilizing DT. The higher accuracy of SIM-A indicates that DF can extract some additional useful information when the time of each cell in the trace is based on the original timestamp. Even so, the benefit of using DT as opposed to without time, can still be seen when using SIM-B.

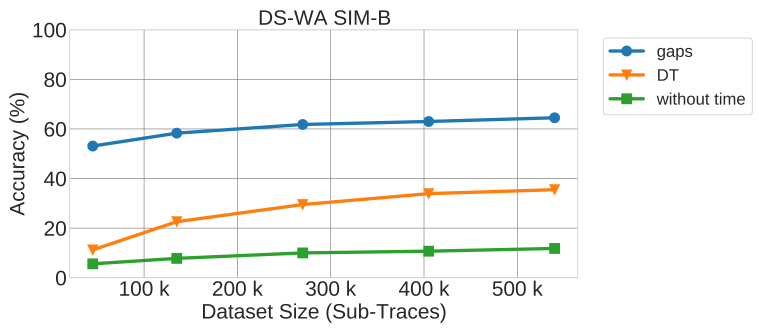

Figure 5.0.3: SIM-B using DS-WA as input.

Figure 5.0.4 shows the accuracy of DF against TrafficSliver using the DS-DF dataset with the SIM-B version of the simulator. The DS-DF dataset does not contain any original timestamps, and can thus not use SIM-A. The accuracy of DF when using only direction (**without time**) goes from 9.8% to 11.5% when increasing the amount of sub-traces. The conceptual gaps of DS-DF (**gaps**) shows an accuracy from 63.6% to 65.7%

for DF when increasing the sub-trace amount. The accuracy for DF when using DT at 475,000 sub-traces is 29.8% (**DT**) compared to 9.8% **without time**. When increasing the amount to sub-traces to three times its original size (1,425,000 sub-traces) the accuracy of DF when using DT ends up at 37.7%. Again, because SIM-B is used, there are no original timestamps and therefore no line for DT and gaps combined.

The accuracy of DF when using DT on DS-DF is relatively constant compared to the other datasets. Give that the size of DS-DF is larger than the other datasets we can conclude that DF is not starved of any data. DS-DF did not contain any original timestamps, which is why SIM-B is used. DF can therefore not extract any additional time data from original timestamps since they do not exist, but the benefit of using a richer data representation can still be seen.

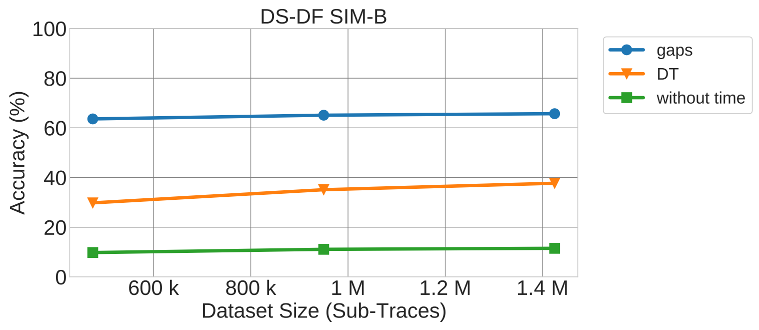

Figure 5.0.4: SIM-B using DS-DF as input.

Chapter 6

Discussion

In this chapter we discuss the results from the previous chapter and then bring up some findings in regards to updating the batch size in BWR. This chapter then concludes with presenting an alternative architecture attempted early on in the study.

6.1 Deep Fingerprinting Classification Accuracy

Table 5.0.1, showing the accuracy for the different non-defended datasets used in this study, shows that DF performed slightly better on DS-TS and DS-DF compared to DS-WA. The slightly lower accuracy on DS-WA could be due to the fact that the dataset contains comparatively fewer cells, which was shown earlier in Table 3.2.1. The roughly 16 million cells in DS-WA is only 40% the amount of DS-TS and 9% the amount of DS-DF.

All of the simulation results show that the performance of DF in regards to accuracy is increased solely by introducing DT to the traces. From 7.1% to 23.1% for DS-TS, from 5.4% to 14.1% for DS-WA SIM-A, from 5.6% to 11.2% for DS-WA SIM-B and from 9.8% to 29.8% for DS-DF. This means that Rahman *et al.* were correct to conclude that WF attacks could perform better with time as additional input. This also means that: *the conceptual gaps introduced in Section 3.1 can be estimated by DF when introducing DT to the traces.*

For DS-TS SIM-A (Figure 5.0.1) and DS-WA SIM-A (Figure 5.0.2) a line was added for the combination of DT and gaps (**DT gaps**). By adding DT to the sub-traces with gaps additional information can be extracted from a burst of cells. When a sub-trace contains a burst of cells, the time difference of said cells might not be equal. This information is something that DF can use to learn more from. In SIM-B no original timestamps are used, this means that the time difference of cells in bursts are equal (based on a small delta-time). The gaps can still be estimated with SIM-B, but some information is lost when not using original timestamps. In the case of DS-DF, these original timestamps were not available anyway.

DS-TS SIM-A (Figure 5.0.1) is the only dataset which has two lines for DT. This was because we had access to the dataset from TrafficSliver containing sub-traces from a real world implementation of BWR5. After having trained DF with sub-traces from the simulator, the real sub-traces were used to test the model. The slight difference in **DT sim** accuracy (testing with simulated sub-traces) and **DT real** accuracy (testing with non-simulated sub-traces) could be from a miss-match between the simulator and the real world implementation or noise from DS-TS. Since DS-TS was created by merging the sub-traces in the dataset from TrafficSliver, the additional real world latency from using multiple guards was integrated into DS-TS. This could mean that the original trace (before the splitting done by TrafficSliver when collecting the dataset) is not a one-to-one match with DS-TS. Both the **DT sim** and **DT real** accuracy with DS-TS are still close to each other, across increasing amounts of sub-traces, demonstrating that: *simulated training data can be used to train DF in order to test on non-simulated data.*

When comparing DS-WA SIM-A (Figure 5.0.2) and DS-WA SIM-B (Figure 5.0.3), the performance of DF when only using direction does not show any large difference, but there is a contrast when using DT. Since both versions of the simulator use the same BWR5 splitting-strategy, the difference here must be affected by the timestamps. SIM-A uses the original timestamps of each cell and replaces the real RTT with samples from a collection of latencies. SIM-B ignores the original timestamp and instead sets the timestamp of the first packet to 1 and adds latency (sampled from the same collection as in SIM-A) and delta-time to subsequent packets to get relative time. The accuracy of DF using SIM-A with DT is closer to the accuracy of the conceptual gaps compared to SIM-B. So when the non-defended dataset contains direction as well as time, the preferable choice is to pick SIM-A. When the non-defended dataset does *not* contain time (DS-DF) the only choice is to pick SIM-B, which still shows that: *the accuracy of DF is increased when using DT, and that this accuracy can then be further increased by using a larger dataset, which can be generated with the simulator.*

DS-DF SIM-B (Figure 5.0.4), does not show the same drastic increase of accuracy when increasing the amount of sub-traces for training. DS-DF is a relatively large dataset (with almost 175 million cells) compared to the other two. This means that DF is not starved of training data with the original size of DS-DF. Here we can see again that the accuracy of DF utilizing DT with a dataset using SIM-B is not as close to the accuracy of conceptual gaps compared to the datasets using SIM-A. But the main observation in Figure 5.0.4 is that: *DT is better than using only direction in DF.*

TrafficSliver tested multiple splitting strategies against DF and found BWR5 to be the best performing one. Conflux on the other hand uses a splitting strategy based on load balancing and congestion awareness which is closer to the purpose of MPTCP. DF has previously achieved a closed world accuracy over 84% against the proposed splitting strategies of MPTCP, which underlines the purpose of performance and not as a defence against WF [10]. Since Conflux is focusing on performance, we expect that it is a worse defence against WF than TrafficSliver, which does not choose a path based on load balancing. The implication of our result against TrafficSliver is that DT and and larger amounts of training data would make DF perform just as well if not better against Conflux.

6.2 Updating Batch Size

A miss-match was found between the description in the TrafficSliver paper [3] of updating the batch size in BWR and the implementations in both the simulator and the Tor implementation. The paper mentions that a vector of weights $\overrightarrow{w}$, is used to select a guard to send a batch of n cells, and states:

> "Thus, we update n constantly during a single page load, i.e., after each batch."

We observed that for both the simulator[1] and the Tor implementation[2] the batch size (variable named C instead of n) is constantly updated *during* each batch and not *after*. In order for the simulator to match the description in the paper, the code would need to be restructured as in Appendix C.1, and the Tor implementation would be restructured as in Appendix C.2.

Changing the BWR splitting-strategy is not within the scope of this study, but could very well be future work to see if this change performs better or worse than the current implementation.

6.3 Alternative Architecture Approach

At the beginning of the project, an alternative approach to data representation was tried and tested. This approach included converting DT to pixel color channels and then use images to leverage a 2-dimensional ResNet model as the CNN architecture. The RGB (Red, Green and Blue) color channel, have 8 bits for each of the three colors to represent a color value, this results in over 16 million values ($2^{24} = 16,777,216$). Since the notion of DT covers both positive and negative integers, the color channel had to be split in two. One half of the color values representing time of incoming cells, and the other half representing time of outgoing cells resulting in over 8 million color values ($2^{24}/2 = 8,388,608$) for each. Additionally, one color value was reserved for the padding and the conceptual gaps. An image would then represent a single trace or sub-trace where each pixel represented the time and direction of each cell (see Figure 6.3.1).

The ResNet model used was implemented using a high level python API for DL called

[1]`https://github.com/TrafficSliver/splitting_simulator/blob/f026aab7dbec944230a6c1ed25538dcd58212947/simulator.py#L215`, accessed 2021-05-25.

[2]`https://github.com/TrafficSliver/trafficsliver-net/blob/b3ac9d2df206b251551d1f27cbca8c4a8fbeb91f/src/feature/split/splitstrategy.c#L645`, accessed 2021-05-25.

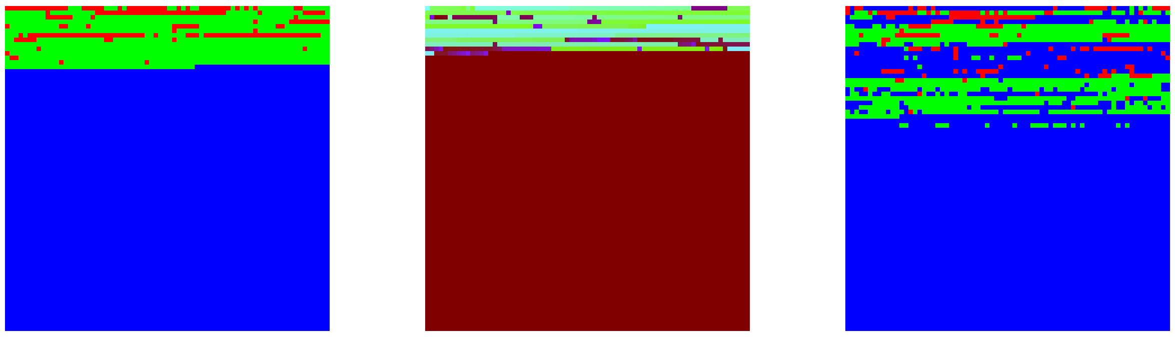
(a) Direction (b) Direction and Time (c) Direction and Gaps

Figure 6.3.1: Images representing traces.

fast.ai[3]. The ResNet, when using DT against TrafficSliver, did not perform as well as DF with DT against TrafficSliver, and was therefore taken off the table for this study. It was after this discovery that the scope of the study was narrowed down to only focus on data representation and keep attack models and architecture outside of the scope. But the idea of using alternative DL architectures (both one and two dimensional CNNs) is still interesting for the area of WF and should be looked more into.

[3]`https://www.fast.ai/`, accessed 2021-05-25.

Chapter 7

Conclusions and Future Work

This chapter summarises the findings from the previous chapter and concludes what implications these results have on adapting a Website Fingerprinting defence for Tor. Then future work is recommended based on the conclusions of this study.

7.1 Conclusion

The goal of this study was to evaluate TrafficSliver as a Website Fingerprinting defence for Tor with the state-of-the-art Website Fingerprinting attack: Deep Fingerprinting. One of the key characteristics of TrafficSliver compared to previous defences was the non-existing overhead, usually comprising of artificial delays or dummy traffic. Results showed that simulated training data could be used to train Deep Fingerprinting against the non-simulated dataset. By utilizing Directional Time as a richer data representation and generating an increased amount of training data from the TrafficSliver splitting-strategy simulator, the accuracy of Deep Fingerprinting was improved. An alternative version of the splitting-strategy simulator from TrafficSliver was implemented which could utilize datasets without original timestamps, such as the dataset collected by Sirinam *et al.* when introducing Deep Fingerprinting.

On the dataset from De la Cadena *et al.* (DS-TS) the accuracy was increased from 7.1% to 49.9%. The accuracy improvement on the dataset from Wang *et al.* (DS-WA) showed an increase from 5.4% to 44.9% with the SIM-A version of the simulator. With the dataset from Sirinam *et al.* (DS-DF) the accuracy was improved from 9.8% to 37.7% with the SIM-B version of the simulator. The results from using SIM-B show that it is possible to utilize datasets without original timestamps with the simulator to generate sub-traces for training Deep Fingerprinting. Comparing SIM-A and SIM-B shows that the accuracy of SIM-A is closer to the conceptual upper limit than the accuracy of SIM-B. This means that datasets containing timestamps should use SIM-A, and that SIM-B should be used when no timestamps are available. Another lesson here is the importance of giving deep learning-attacks enough data for training, especially if probabilistic defences are evaluated.

Given the new results of Deep Fingerprinting against TrafficSliver we conclude that traffic splitting is not as strong of a Website Fingerprinting defence as previously thought. One fundamental assumption of traffic splitting, without introducing any network overhead such as dummy traffic and artificial delays, is that an attacker is only able to observe a subset of the user's total traffic, and that traffic splitting could therefore work as a Website Fingerprinting defence. When introducing Directional Time as a richer data representation this assumption no longer holds, since Website Fingerprinting attacks could estimate the timing and magnitude of "unobserved" traffic. Tor Proposal 329 is focusing on using traffic splitting for increasing performance in the Tor network and performance-based splitting-strategies has been proved a weak defence against Deep Fingerprinting. That is why other types of defences should also be included in future studies around Website Fingerprinting in Tor.

7.2 Future Work

This study did not collect any new dataset from the live Tor network, but utilized previously collected datasets. Future work would include collecting a new *larger* dataset with TrafficSliver (not using simulation) and use it to train Deep Fingerprinting against the defence. An alternative architecture approach was discarded early on in the project in order to focus on data representation (Directional Time) and ways to increase the training dataset (splitting-strategy simulator). Future work for the field of Website Fingerprinting include looking for architectures which could outperform Deep Fingerprinting. The scope of this study was to only explore the closed world setting in Website Fingerprinting. It is important to evaluate attacks and defences in a bigger context such as the open world setting since all trade-offs are not clear in the closed world setting. It is therefore a good idea to investigate the open world setting using Directional Time in future work. As described in Section 6.2 we found a miss-match between the description in the TrafficSliver paper of updating the batch size in BWR and the implementations in both the simulator and the Tor implementation. One suggestion of future work is to see if this change performs differently than the current implementation.

Bibliography

[1] AlSabah, Mashael, Bauer, Kevin S., Elahi, Tariq, and Goldberg, Ian. “The Path Less Travelled: Overcoming Tor’s Bottlenecks with Traffic Splitting”. In: *Privacy Enhancing Technologies - 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings*. Ed. by Emiliano De Cristofaro and Matthew K. Wright. Vol. 7981. Lecture Notes in Computer Science. Springer, 2013, pp. 143–163. DOI: `10.1007/978-3-642-39077-7\_8`. URL: `https://doi.org/10.1007/978-3-642-39077-7%5C_8`.

[2] Bojarski, Mariusz, Testa, Davide Del, Dworakowski, Daniel, Firner, Bernhard, Flepp, Beat, Goyal, Prasoon, Jackel, Lawrence D., Monfort, Mathew, Muller, Urs, Zhang, Jiakai, Zhang, Xin, Zhao, Jake, and Zieba, Karol. “End to End Learning for Self-Driving Cars”. In: *CoRR* abs/1604.07316 (2016). arXiv: `1604.07316`. URL: `http://arxiv.org/abs/1604.07316`.

[3] Cadena, Wladimir De la, Mitseva, Asya, Hiller, Jens, Pennekamp, Jan, Reuter, Sebastian, Filter, Julian, Engel, Thomas, Wehrle, Klaus, and Panchenko, Andriy. “TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting”. In: *CCS ’20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM, 2020, pp. 1971–1985. DOI: `10.1145/3372297.3423351`. URL: `https://doi.org/10.1145/3372297.3423351`.

[4] Das, Debajyoti, Meiser, Sebastian, Mohammadi, Esfandiar, and Kate, Aniket. “Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency - Choose Two”. In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 108–126. DOI: `10.1109/SP.2018.00011`. URL: `https://doi.org/10.1109/SP.2018.00011`.

[5] Dingledine, Roger, Mathewson, Nick, and Syverson, Paul F. “Tor: The Second-Generation Onion Router”. In: *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*. Ed. by Matt Blaze. USENIX, 2004, pp. 303–320. URL: `http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html`.

[6] Frigyik, Bela A, Kapila, Amol, and Gupta, Maya R. “Introduction to the Dirichlet distribution and related processes”. In: *Department of Electrical Engineering, University of Washignton, UWEETR-2010-0006* 0006 (2010), pp. 1–27.

[7] Goldberg, Ian, Wagner, David A., and Brewer, Eric A. "Privacy-enhancing technologies for the Internet". In: *Proceedings IEEE COMPCON 97, San Jose, California, USA, February 23-26, 1997, Digest of Papers*. IEEE Computer Society, 1997, pp. 103–109. DOI: 10.1109/CMPCON.1997.584680. URL: https://doi.org/10.1109/CMPCON.1997.584680.

[8] Goldschlag, David M., Reed, Michael G., and Syverson, Paul F. "Onion Routing". In: *Commun. ACM* 42.2 (1999), pp. 39–41. DOI: 10.1145/293411.293443. URL: https://doi.org/10.1145/293411.293443.

[9] Hayes, Jamie and Danezis, George. "k-fingerprinting: A Robust Scalable Website Fingerprinting Technique". In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, 2016, pp. 1187–1203. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes.

[10] Henri, Sébastien, Garcia-Aviles, Gines, Serrano, Pablo, Banchs, Albert, and Thiran, Patrick. "Protecting against Website Fingerprinting with Multihoming". In: *Proc. Priv. Enhancing Technol.* 2020.2 (2020), pp. 89–110. DOI: 10.2478/popets-2020-0019. URL: https://doi.org/10.2478/popets-2020-0019.

[11] Hintz, Andrew. "Fingerprinting Websites Using Traffic Analysis". In: *Privacy Enhancing Technologies, Second International Workshop, PET 2002, San Francisco, CA, USA, April 14-15, 2002, Revised Papers*. Ed. by Roger Dingledine and Paul F. Syverson. Vol. 2482. Lecture Notes in Computer Science. Springer, 2002, pp. 171–178. DOI: 10.1007/3-540-36467-6_13. URL: https://doi.org/10.1007/3-540-36467-6%5C_13.

[12] Jansen, Rob, Juárez, Marc, Galvez, Rafa, Elahi, Tariq, and Díaz, Claudia. "Inside Job: Applying Traffic Analysis to Measure Tor from Within". In: *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018%5C_10-2%5C_Jansen%5C_paper.pdf.

[13] Juárez, Marc, Afroz, Sadia, Acar, Gunes, Díaz, Claudia, and Greenstadt, Rachel. "A Critical Evaluation of Website Fingerprinting Attacks". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM, 2014, pp. 263–274. DOI: 10.1145/2660267.2660368. URL: https://doi.org/10.1145/2660267.2660368.

[14] Juárez, Marc, Imani, Mohsen, Perry, Mike, Díaz, Claudia, and Wright, Matthew. "WTF-PAD: Toward an Efficient Website Fingerprinting Defense for Tor". In: *CoRR* abs/1512.00524 (2015). arXiv: 1512.00524. URL: http://arxiv.org/abs/1512.00524.

[15] LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey E. “Deep learning”. In: *Nat.* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[16] Li, Qing, Cai, Weidong, Wang, Xiaogang, Zhou, Yun, Feng, David Dagan, and Chen, Mei. “Medical image classification with convolutional neural network”. In: *13th International Conference on Control Automation Robotics & Vision, ICARCV 2014, Singapore, December 10-12, 2014*. IEEE, 2014, pp. 844–848. DOI: 10.1109/ICARCV.2014.7064414. URL: https://doi.org/10.1109/ICARCV.2014.7064414.

[17] Milioto, Andres, Lottes, Philipp, and Stachniss, Cyrill. “Real-Time Semantic Segmentation of Crop and Weed for Precision Agriculture Robots Leveraging Background Knowledge in CNNs”. In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 2229–2235. DOI: 10.1109/ICRA.2018.8460962. URL: https://doi.org/10.1109/ICRA.2018.8460962.

[18] Mitchell, Tom M et al. “Machine learning”. In: (1997).

[19] Oh, Se Eun, Sunkam, Saikrishna, and Hopper, Nicholas. “p1-FP: Extraction, Classification, and Prediction of Website Fingerprints with Deep Learning”. In: *Proc. Priv. Enhancing Technol.* 2019.3 (2019), pp. 191–209. DOI: 10.2478/popets-2019-0043. URL: https://doi.org/10.2478/popets-2019-0043.

[20] Panchenko, Andriy, Lanze, Fabian, Pennekamp, Jan, Engel, Thomas, Zinnen, Andreas, Henze, Martin, and Wehrle, Klaus. “Website Fingerprinting at Internet Scale”. In: *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/website-fingerprinting-internet-scale.pdf.

[21] Panchenko, Andriy, Niessen, Lukas, Zinnen, Andreas, and Engel, Thomas. “Website fingerprinting in onion routing based anonymization networks”. In: *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society, WPES 2011, Chicago, IL, USA, October 17, 2011*. Ed. by Yan Chen and Jaideep Vaidya. ACM, 2011, pp. 103–114. DOI: 10.1145/2046556.2046570. URL: https://doi.org/10.1145/2046556.2046570.

[22] Rahman, Mohammad Saidur, Sirinam, Payap, Mathews, Nate, Gangadhara, Kantha Girish, and Wright, Matthew. “Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks”. In: *Proc. Priv. Enhancing Technol.* 2020.3 (2020), pp. 5–24. DOI: 10.2478/popets-2020-0043. URL: https://doi.org/10.2478/popets-2020-0043.

[23] Rimmer, Vera, Preuveneers, Davy, Juárez, Marc, Goethem, Tom van, and Joosen, Wouter. “Automated Website Fingerprinting through Deep Learning”. In: *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society,

2018. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018%5C_03A-1%5C_Rimmer%5C_paper.pdf.

[24] Sirinam, Payap, Imani, Mohsen, Juárez, Marc, and Wright, Matthew. "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM, 2018, pp. 1928–1943. DOI: 10.1145/3243734.3243768. URL: https://doi.org/10.1145/3243734.3243768.

[25] Wang, Tao, Cai, Xiang, Nithyanand, Rishab, Johnson, Rob, and Goldberg, Ian. "Effective Attacks and Provable Defenses for Website Fingerprinting". In: *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*. Ed. by Kevin Fu and Jaeyeon Jung. USENIX Association, 2014, pp. 143–157. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/wang%5C_tao.

[26] Wang, Tao and Goldberg, Ian. "Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks". In: *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*. Ed. by Engin Kirda and Thomas Ristenpart. USENIX Association, 2017, pp. 1375–1390. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-tao.

Appendix - Contents

Appendix A

Simulation Code

```
def simulate_A(instance,mplatencies,routes,seed):
    random.seed(seed)
    # Delta time to introduce as the time between
    # two cells are sent from the end side
    delta = 0.0001
    delta *= 10**6 # convert to microseconds

    last_packet = 1
    last_time = 0
    delay = 0
    time_last_incomming = 0
    new_trace = []
    for i in range(len(instance)):
        last_incomming_cell = 0
        packet = instance[i]
        original_time = abs(packet)
        direction = 1 if int(packet) > 0 else -1
        size = 512
        route = routes[i]
        chosen_latency = float(random.choice(mplatencies[(route%len(
   mplatencies))]))
        chosen_latency *= 10**6 # convert to microseconds

        if (i != 0 and last_packet != direction):
            delay = float(original_time - last_time)/2

        new_packet = np.array([])

        if (direction == -1):
            new_packet=[original_time - delay + chosen_latency,direction,
   size,route]
        if (direction == 1 and last_packet == -1):
            # If is the first out in the burst, it referes to the last
   incoming time
            new_packet=[time_last_incomming + delta,direction,size,route]
        if (direction == 1 and last_packet == 1):
            # If we are in an out burst, refers to the last out
            new_packet=[last_time + delta,direction,size,route]
        if i == 0:
            # JM: Added an exception for the first packet
            new_packet=[original_time,direction,size,route]

        new_trace.append(new_packet)
        time_last_incomming = original_time - delay + chosen_latency
        last_time = original_time
        last_packet = direction
    np_new_trace =  np.array(new_trace)
    sorted_new_trace = np_new_trace[np_new_trace[:,0].argsort()]
    #Sorted according to the new timestamps

    return sorted_new_trace
```

Listing A.1: Version A of the simulation (using original time).

```
def simulate_B(instance,mplatencies,routes,seed):
    random.seed(seed)
    # Delta time to introduce as the time between
    # two cells are sent from the end side
    delta = 0.0001
    delta *= 10**6 # convert to microseconds

    last_direction = 1
    last_time = 0
    new_trace = []
    for i in range(len(instance)):
        if int(instance[i]) != 0: # DS-DF pads with zeroes at the end
            direction = 1 if int(instance[i]) > 0 else -1
            route = routes[i]
            chosen_latency = float(random.choice(mplatencies[(route%len(
   mplatencies))]))
            chosen_latency *= 10**6 # convert to microseconds

            # If we were sending and just received something new,
            # assume it took chosen_latency time to deliver
            if (direction == -1 and last_direction == 1):
                last_time += chosen_latency
            else:
                last_time += delta

            new_trace.append(np.array([last_time,direction,512,route]))
            last_direction = direction

    np_new_trace =  np.array(new_trace)
    sorted_new_trace = np_new_trace[np_new_trace[:,0].argsort()]
    #Sorted according to the new timestamps

    return sorted_new_trace
```

Listing A.2: Version B of the simulation (not using original time).

Appendix B

Deep Fingerprinting Implementation Code

```python
def split_dataset_BWR5(classes, sample_tracker, sub_traces):
    # Splits the dataset based on IDs, not the actual data.
    # The result is a 8:1:1 split into
    # training, validation, and testing.

    training = []
    validation = []
    testing = []

    # monitored, split by sample
    for c in range(0,classes):
        for s in range(0,sample_tracker[c]):
            x = (s % sub_traces) // 5
            ID = f"m-{c}-0-{s}"
            if x < int(sub_traces/5*0.8):
                training.append(ID)
            elif x < int(sub_traces/5*0.9):
                validation.append(ID)
            else:
                testing.append(ID)

    split = {}
    split["train"] = training
    split["validation"] = validation
    split["test"] = testing
    return split
```

Listing B.1: Splitting function for the BWR5 datasets.

Appendix C

Alternative Batch-Size Code

```
# First routes are weighted random chosen
last_client_route =  np.random.choice(np.arange(0,n),p = w_out)
last_server_route = np.random.choice(np.arange(0,n),p = w_in)
# After how many cells the scheduler sets new weights
C = random.randint(ranlow,ranhigh)

for i in xrange(0,len(instance)):
    packet = instance[i]
    packet = packet.replace(' ','\t')
    direction = multipath.getDirfromPacket(packet)

    if (direction == 1):
        routes_server.append(-1)
        sent_outgoing += 1
        routes_client.append(last_client_route)
        # After C cells are sent, change the circuits
        if (sent_outgoing % C == 0):
            last_client_route = np.random.choice(np.arange(0,n),p = w_out)
            C = random.randint(ranlow,ranhigh)

    if (direction == -1):
        routes_client.append(-1)
        routes_server.append(last_server_route)
        sent_incomming += 1
        # After C cells are sent, change the circuits
        if (sent_incomming % C == 0):
            last_server_route = np.random.choice(np.arange(0,n),p = w_in)
            C = random.randint(ranlow,ranhigh)
```

Listing C.1: Restructuring of updating batch size for the simulator.

```
// First route is weighted random chosen
current_id =  weighted_paths[crypto_rand_int_range(0,100)];
// After how many cells the scheduler sets new weights
int current_batch_size = crypto_rand_int_range(C_MIN, C_MAX);

for (int pos = 0; pos < num; pos++) {
    do {
        if ((pos % current_batch_size)==0){
            // After the batch size, perform a new weighted random choice
            current_id = weighted_paths[crypto_rand_int_range(0,100)];
            current_batch_size = crypto_rand_int_range(C_MIN, C_MAX);
        }
    } while (!subcirc_list_get(subcircs, current_id));

    write_subcirc_id(current_id, list + pos);
}
```

Listing C.2: Restructuring of updating batch size for the Tor implementation.