Splitting Hairs and Network Traces

Improved Attacks Against Traffic Splitting as a Website Fingerprinting Defense

Matthias Beckerle
Karlstad University
Sweden
matthias.beckerle@kau.se

Jonathan Magnusson
Karlstad University
Sweden
jonathan.magnusson@kau.se

Tobias Pulls
Karlstad University
Sweden
tobias.pulls@kau.se

ABSTRACT

The widespread use of encryption and anonymization technologies—e.g., HTTPS, VPNs, Tor, and iCloud Private Relay—makes network attackers likely to resort to traffic analysis to learn of client activity. For web traffic, such analysis of encrypted traffic is referred to as Website Fingerprinting (WF). WF attacks have improved greatly in large parts thanks to advancements in Deep Learning (DL). In 2019, a new category of defenses was proposed: traffic splitting, where traffic from the client is split over two or more network paths with the assumption that some paths are unobservable by the attacker.

In this paper, we take a look at three recently proposed defenses based on traffic splitting: HyWF, CoMPS, and TrafficSliver BWR5. We analyze real-world and simulated datasets for all three defenses to better understand their splitting strategies and effectiveness as defenses. Using our improved DL attack *Maturesc* on real-world datasets, we improve the classification accuracy wrt. state-of-the-art from 49.2% to 66.7% for HyWF, the F_1 score from 32.9% to 72.4% for CoMPS, and the accuracy from 8.07% to 53.8% for TrafficSliver BWR5. We find that a majority of wrongly classified traces contain less than a couple hundred of packets/cells: e.g., in every dataset 25% of traces contain less than 155 packets. What cannot be observed cannot be classified. Our results show that the proposed traffic splitting defenses on average provide less protection against WF attacks than simply randomly selecting one path and sending all traffic over that path.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Networks → Network privacy and anonymity**.

KEYWORDS

website fingerprinting; deep learning; network splitting

ACM Reference Format:
Matthias Beckerle, Jonathan Magnusson, and Tobias Pulls. 2022. Splitting Hairs and Network Traces: Improved Attacks Against Traffic Splitting as a Website Fingerprinting Defense. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society (WPES '22), November 7, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3559613.3563199

WPES '22, November 7, 2022, Los Angeles, CA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9873-2/22/11.
https://doi.org/10.1145/3559613.3563199

1 INTRODUCTION

The web has widely adopted HTTPS, with a majority of connections today being encrypted [15]. The DNS is also slowly evolving, seeing increased use of encrypted connections using, e.g., DNS-over-HTTPS (DoH) [26], DNS-over-TLS (DoT) [29], and DNS-over-QUIC (DoQ) [30]. In parallel, consumers increasingly use technologies like VPNs [2, 51] and the anonymity network Tor [11] to protect their activities. Recently, Apple announced iCloud Private Relay [5], a VPN-like technology that can be seen as a middle-ground between a VPN and Tor. With content (HTTPS and Do{H,T,Q}) and destination IP-addresses (VPNs, Tor, iCloud Private Relay) hidden, attackers observing network traffic from clients are increasingly limited in their visibility of client activity. One remaining avenue is analyzing patters in the encrypted traffic.

In the Website Fingerprinting (WF) setting, a local passive attacker analyzes patterns in encrypted traffic to fingerprint websites visited by a victim client [8, 24, 25, 39, 58]. The attacker is "local" in the sense that they are close to the client: it could be, e.g., the client's Internet Service Provider (ISP), a WiFi hotspot, or the client's mobile service provider. The attacker is "passive" in the sense that they only eavesdrop on network traffic without interfering with the communication. WF attacks typically target the frontpage of websites, i.e., the goal of the attacker is to determine if a victim has visited a particular website, instead of particular webpages as part of a website. A concrete example would be to fingerprint visits to wikipedia.org instead of visits to, e.g., the English Wikipedia page on the origin of "Slava Ukraini" at en.wikipedia.org/wiki/Slava_Ukraini.

WF attacks, just like for most attacks in general, have become increasingly more powerful over the years. The community spent many years on perfecting manual feature engineering—which aspects of encrypted network traces to consider—to improve WF attack classification accuracy when using traditional machine learning techniques like k-nearest neighbors [62] and support vector machines [45]. Around 2016, k-fingerprinting was the state-of-the-art WF attack based on manual features [20] and the community was moving towards promising efficient and effective defenses [35]. Unfortunately, shortly thereafter the massive advances in Deep Learning (as part of the ongoing "AI spring") enabled automatic feature engineering—just let the neural network learn features from the raw data—that vastly surpassed the manual features. Consequently, WF attacks based on Deep Learning (DL) are successful against many state-of-the-art WF defenses [6, 50, 55]. More broadly, the last five years have seen massive advances in the area of DL. Combined with advances in available hardware, DL gets faster, more precise and better understood, enabling new architectures and learning strategies, ultimately leading to better attacks [22].

WF defenses take different forms and have different trade-offs with a history as long as that of WF attacks [27]. In this paper, we focus on a category of WF defenses that was first proposed in the context of WF in 2019 [37] and has shown promising effectiveness also against WF attacks utilizing DL: traffic splitting. In traffic splitting, the setting of WF attacks is relaxed to assume that a WF attacker can only observe one out of many *paths* used by the client to communicate. Figure 1 shows three such possible settings for traffic splitting in Tor.

Henri et al., for their WF defense HyWF [23], propose that clients use multihoming (multiple Internet connections) to split their traffic to the same guard[1] over *two* paths (Figure 1a). De la Cadena et al. propose TrafficSliver BWR [36], where traffic is split over multiple paths to the same middle (Figure 1b). The most effective version of TrafficSliver is BWR5, where traffic is split over *five* paths (only two paths are shown in Figure 1b). Figure 1c shows Conflux by AlSabah et al. [4], where traffic is split over *two* paths all the way to the exit. Conflux was proposed already in 2013, but as a performance optimization for bridge users. The Tor Project is investigating traffic splitting inspired by Conflux [18]. Beyond Tor, Wang et al. [60] recently proposed the CoMPS framework for any service supporting connection migration (e.g., QUIC [32] and WireGuard [12]), using *three* paths to connect to a server supporting connection migration, e.g., QUIC over multiple WireGuard VPNs.

In this paper, we evaluate WF defenses based on traffic splitting by utilizing a newly created WF attack that based on recent developments in DL. Our contributions are:

- We analyze three traffic splitting WF defenses—HyWF [23] using two paths, CoMPS [60] using three paths, and TrafficSliver BWR5 [36] using five paths—in terms of their designs, splitting strategies, and resulting datasets from both real-world implementations and simulation. Our analysis shows that all three defenses have similar splitting strategies, and that for all datasets 25% of the traces contain less than 155 packets/cells. This poses a significant challenge for any WF attacker, because it is deprived (by assumption in the relaxed setting) of the vast majority of traffic, lowering the accuracy one should reasonably expect from defenses and attacks in this setting (Section 3).
- Using developments in the rapidly evolving DL area, we provide improved attacks that refine the evaluated effectiveness of the analyzed traffic splitting defenses (Section 4). On respective authors' real-world datasets, we improve the state-of-the-art accuracy from 49.2% to 66.7% for HyWF, the F_1 score from 32.9% to 72.4% for CoMPS, and the accuracy from 8.1% to 53.8% for TrafficSliver BWR5. We accomplish this by utilizing xresnets [56], one-cycle training [22], and extensive hyperparameter tuning. Our attack also shows competitive accuracy on regular (non-split) WF tasks (Appendix A).
- Based on our findings, we further analyze the relaxed traffic splitting setting as a WF defense (Section 5). We conclude that traffic splitting across multiple paths increases the threat posed by WF attacks, because a WF attacker will be highly successful once it can observe a couple of hundred of packets/cells. The main defense from traffic splitting is provided by the underlying assumption that an attacker may not be able to observe nearly *any* traffic, which is not the case against many relevant threat actors within the threat model of Tor [11]. When attackers can observe traffic other WF defenses are needed, and therefore traffic splitting should be viewed mainly as a performance feature. Luckily, this aligns well with how scheduling algorithms like BLEST and minRTT behave [16, 31, 40] and the proposal from the Tor Project to investigate such algorithms [18].

[1]Technically, they propose using a multipath bridge, but it would serve the same purpose as the guard.

Section 2 covers necessary background, Sections 3–5 contributions, Section 6 related work, and Section 7 conclusions.

2 BACKGROUND

We briefly cover relevant background on Tor, Website Fingerprinting, and Deep Learning.

2.1 Tor

Tor is a low-latency anonymity network with millions of daily users that use Tor mainly to browse the web anonymously, access so-called onion services, and circumvent censorship [11]. The Tor network consists of more than 7,000 volunteer-run servers, called *relays*. Connections in Tor are made through *circuits*, layered encrypted connections that traverse three or more relays in the network. Within circuits, data is transported in fixed-sized cells of 514 bytes that are encrypted both on the cell-level and network-level (using TLS between relays). For a regular (non-onion) circuit, the first relay is called the entry guard relay, the second relay the middle relay, and the third final relay the exit relay. The relays get their respective names from the role they play, where notably users will use many different middle and exit relays, but restrict their guards to a small set of relays. The *guard set* of relays are selected by users once and kept the same for months, motivated by a wide range of security concerns [19]. At the time of writing, the size of the set is configured to 1.

Tor adds minimal latency and bandwidth overheads, and is not designed to provide anonymity against an attacker that can observe all traffic in the Tor network [11]. There are well known trade-offs in terms of latency, bandwidth, and provided anonymity [9, 10], but Tor strongly favours performance with the goal of attracting more users. In general, the weaker the attacker, the more concerned Tor is with deploying defenses against attacks by such attackers.

2.2 Website Fingerprinting

A type of attack that falls directly within the threat model of Tor is a Website Fingerprinting (WF) attack. The setting of a WF attack involves a local passive attacker that observes encrypted network traffic (a trace) between a target user and their guard, with the goal of determining (classifying) the website (typically frontpage of the website) the user is visiting. Such an attacker could be the WiFi-network the user is connected to, their Internet Service Provider (ISP), any intermediate Autonomous System (AS), middleboxes, nation-state actors, or even the guard itself.

WF attacks and defenses are evaluated in either a "closed" or "open" world. In the closed world, the goal of the attacker is to

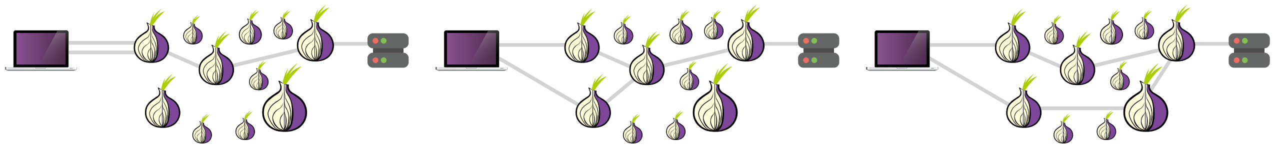
(a) Split to guard, HyWF [23] (b) Split to middle, TrafficSliver BWR [36] (c) Split to exit, Conflux [4]

Figure 1: Three approaches to traffic splitting in Tor.

identify which website a trace belongs to out of a fixed number of possible *monitored* websites (typically on the order of a 100 websites). In the open world, the attacker has the same goal as in the closed world, but some traces may now come from *unmonitored* websites: conceptually every other possible website on the web except for the monitored websites. The open world is considered more realistic, but opens up issues such as the base rate of monitored websites in datasets: typically datasets contain 50% monitored and 50% unmonitored traces, but this is not a representative split [34, 47]. The closed world, while less realistic, is useful for comparison between attacks and defenses.

2.2.1 Attacks. WF attacks have a long history, and before mainly being focused on Tor considered HTTPS [8], web proxies [25, 58], SSH tunnels [39], and VPNs [24]. Central to attacks are features extracted from traces for use in classification, and it is possible to group attacks based on manual or automatic feature engineering.

With manual feature engineering, attackers define their own features. Known important features are, e.g., the number of incoming packets and the fraction of incoming/outgoing packets. The state of the art WF attack in this category is k-fingerprinting by Hayes and Danezis [20], that uses a random forest classifier with 150 carefully selected features (cf., e.g., Wang et al.'s earlier and less effective kNN attack with over 3,000 features [62]).

The advent of deep learning (see Section 2.3) brought with it automatic feature extraction for attacks [1, 52]. Attackers now consider different representations of traces as input data, and the neural networks automatically extract latent feature spaces with richer information than any manually engineered feature set. The two main representations of input data are as *directional sequences* or *directional time*. For directional sequences, the data consists just of a sequence of integers denoting packet or cell sizes where positive or negative values indicating sent or received. Because Tor sends all data as padded 514-byte cells, traces for Tor typically consists of a sequence of +1 and -1. Directional time is a data representation for Tor where the time a cell was sent is represented either as a positive or negative float, depending on direction.

State of the art WF attacks using deep learning (in terms of maximum accuracy, not considering training time or the amount of available training) are Deep Fingerprinting (DF) by Sirinam et al. [55], Var-CNN by Bhat et al. [6], and Tik-Tok by Rahman et al. [50]. DF is a ResNet-inspired [21] architecture that notably uses directional sequences (without time), yet has excellent performance, including on defenses such as WTF-PAD [35]. Tik-Tok improves DF by changing the data representation to instead use directional time. Var-CNN is an ensemble classifier based on a ResNet-18 architecture using dilated causal convolutions, where one classifier uses directional sequences and another direction time combined with cumulative statistical features of the network trace.

2.2.2 Defenses. Broadly, most WF defenses can be categorized into four categories: imitation, regulation, alteration, and traffic splitting [27]. Imitation defenses, like Glove [43] and Walkie-Talkie [64], aim to make traces look similar to that of other known traces. This requires a database of known traces (of key features of those traces) to target. Such a database is a practical challenge, and newer defenses like Walkie-Talkie that make trade-offs to become more practical have been shown to be vulnerable to deep learning attacks like Tik-Tok.

Regulation defenses, like RegulaTor [27], Tamaraw [7], and WTF-PAD [35], aim to make all traces look similar. Provably secure defenses like Tamaraw impose significant latency and bandwidth overheads. Less strict defenses, pioneered by WTF-PAD, relax the security guarantees to reduce overheads. WTF-PAD showed great promise against WF attacks using classical machine learning, but has been shown to be vulnerable to deep learning based attacks such a DF [55]. Regulator is a new defense in this category that manages low overhead and a significant challenge for deep learning based attacks [27].

Alteration defenses, like HTTPOS [41] and FRONT [17], aim to change traces to such an extent that attackers cannot find any useful patterns. As a primarily example, FRONT focuses on heavily padding the early (front) parts of traces achieving modest overheads while reducing accuracy also for deep learning based attacks [17].

Finally, traffic splitting defenses, like HyWF [23], CoMPS [60], and TrafficSliver BWR5 [36], aim to split traces over two or more network paths, where the attacker is assumed to only be able to observe one path. This is a new approach to WF defenses, and the category we focus on in this paper. Section 3 provides further details.

2.3 Deep Learning

Deep Learning (DL) is a form of machine learning that mimics biological learning by utilizing neural networks with more than three hidden layers. DL is usually outperforming classical approaches of machine learning and reaches super human levels in multiple domains, e.g., in classification problems. It has the additional advantage that no feature engineering is needed. It works best when raw data is used as input, which is convenient and efficient.

DL and machine learning in general face common challenges. For a successful training and application of the model the following challenges have to be meet:

- **Physical limitations:** Especially the amount of GPU memory can limit the model size and input amount. The speed of the CPU and GPU determine learning speed.
- **Data limitations:** There are several factors regarding the data that can limit learning success. These factors are: too little training data, noisy data (e.g., if it is defended), mislabeled data, and non-representative data (that does not reflect reality).
- **Hyperparameter tuning:** Hyperparameters have to be chosen correctly for optimal training results. For more details see Section 2.3.1.

There are several building blocks for successful DL. The correct data preparation, the correct DL architecture, and a fitting set of hyperparameters have to be chosen.

It is important to choose an architecture that fits the problem. In the context of WF, convolutional networks (ConvNets [38]) or the successor residual nets (ResNets [21]) are commonly used [1, 6, 52, 55]. While originally developed for other use cases they work well for WF too.

2.3.1 Hyperparameters. In the context of ML, hyperparameters are the tunable parameters that influence and control the learning process. We considered eight hyperparameters in three categories:

- Two related to the data: trace length and input channels.
- Two related to the architecture: number of layers and dropout in %.
- Four related to the training of the model: learning rate, momentum, batch size, and epochs.

To find the right set of hyperparameters, it is often most efficient to start with common best practises and adjust after reviewing initial results. Hardware limitations may also limit the range of possible hyperparameters. Especially the amount of input data per instance, the complexity of the architecture, and the chosen batch size increase the memory footprint proportionally. However, more does not always mean better, allowing state of the art results even on modest hardware.

2.3.2 One-Cycle Training. Instead of static learning rates and static momentum, it is beneficial to change these parameters while the training is progressing as described by Smith [56]. The learning rate of One-cycle training is the maximum learning rate that is reached in one-cycle training. Momentum is not manually tuned since it is automatically adjusted based on the learning rate. How learning rate and momentum change during the training process can be seen in Figure 2. If used correctly, One-cycle training can drastically reduce training times and even improve the accuracy of a model.

3 UNDERSTANDING TRAFFIC SPLITTING

We first look at the three most effective splitting strategies in detail and then analyze the resulting network traces.

3.1 Splitting Strategies

Figure 3 shows the three splitting strategies we consider due to their assessed strengths as WF defenses:

- HyWF by Henri et al. [23], their strongest defense that splits traffic over two network paths (Figure 3a).

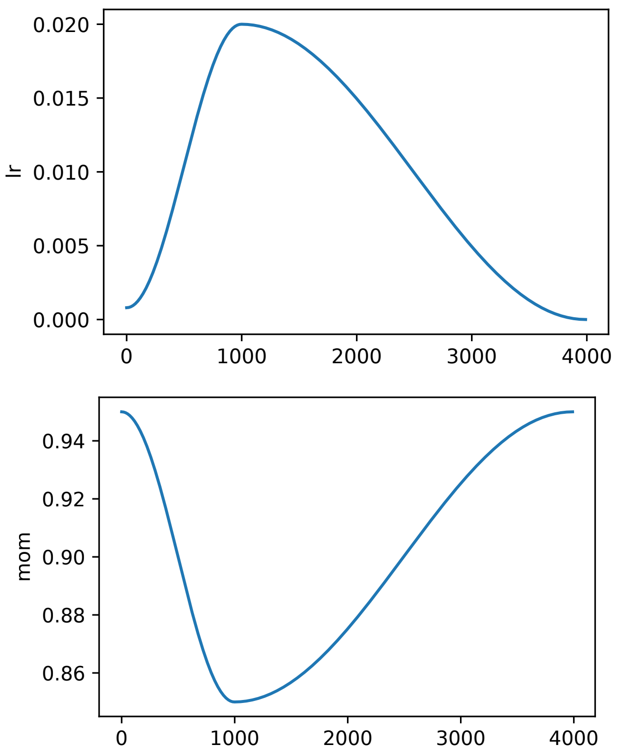

Figure 2: An example of how learning rate (lr) on the left and momentum (mom) on the right changes over time in one-cycle training. The x-axes shows the amount of training batches. This example is from our training on the comps-rw closed dataset with optimized hyperparameters. The detailed parameters can be seen in the first data-row in Table 2.

- The Batched Weighted Random (BWR) strategy of Traffic-Sliver by De la Cadena et al. [36], where traffic is split over five network paths (Figure 3b).
- The Weighted Random (WR) strategy of CoMPS by Wang et al. [60] that splits traffic over three network paths (Figure 3c).

The three strategies are similar, but differ in the number of network paths they use. Note that the CoMPS design was motivated by ease of implementation, based on BWR from TrafficSliver. For sake of comparison, we go over all three strategies in unison.

First, each strategy assigns a random weight to their respective network paths (line 1). The weights are sampled once per website visit to defend. Next, each strategy assigns initial zero values to variables (line 2) and starts iterating over packets to send (line 3). Each strategy updates the network path to send packets over once a threshold is reached. For HyWF and BWR, the threshold is based on the number of packets sent since the last update (lines 4–5). For CoMPS, this is based on elapsed time since the last update (line 4). The selection of path to send packets on is sampled based on the weights assigned earlier to each path ($p \xleftarrow{\$} w$). For HyWF and BWR, the number of packets to send before the next update is sampled from their respective distributions (line 7), while the duration between updates in CoMPS is fixed to 100 ms (line 4).

Table 1: Structure of the ten datasets. The total for the simulated datasets (closed world) comes from `sites × inst + sites × inst × p × x`, where `sites × inst` is the original Wang k-NN dataset size (traces available only for *training*) and `sites × inst × p × x` from simulating `p` paths `x` times. Simulated open world is twice the size of closed world.

dataset	sites	inst	unmon	p	x	total
comps-rw-closed	107	100		1	1	10700
comps-rw-open	107	100	10324	1	1	21024
hywf-rw	100	100		2	1	20000
ts-bwr5-rw	100	100		5	1	48574
comps-sim-closed	100	90		3	10	279000
hywf-sim-closed	100	90		2	10	189000
ts-bwr5-sim-closed	100	90		5	10	459000
comps-sim-open	100	90	9000	3	10	558000
hywf-sim-open	100	90	9000	2	10	378000
ts-bwr5-sim-open	100	90	9000	5	10	918000

These are the main differences between the three strategies, in addition to their varying number of paths.

3.2 Dataset Analysis

To better understand the splitting strategies, we look at the resulting datasets of traces from their use. For each strategy, we analyze two types of datasets: a real dataset provided by the authors of the strategy ("rw", short for real-world) and a simulated dataset from our own simulation ("sim") of the strategy. The real datasets are research artefacts provided by the respective authors and that were collected from the implementations of their strategies. The simulated datasets are based on the widely used Wang k-NN dataset [62]. Simulating each strategy from the same underlying dataset allows for a more accurate comparison.

Table 1 summarizes the ten datasets we use in this paper. We consider both open and closed worlds where data is available. In general, unless otherwise stated, assume a closed world. Noteworthy details on the datasets:

- `comps-rw-open` only contains a single path (the one observed by the WF attacker) and is slightly unbalanced between monitored/unmonitored.
- `ts-bwr5-rw` is missing $\approx$ 3% of traces due to being unspecified in the original dataset (artefact of data format and splitting strategy). Other traces in all other datasets are also of length zero (see later analysis).
- We repeatedly simulate the splitting strategies ten times per original trace, providing ample more data for training, as was shown useful for TS BWR5 [42] and HyWF [23].
- All datasets contain timestamps, but as see later in Section 4.3, time is not useful in the `hywf-rw` dataset.

Figure 4 shows the distributions as boxen plots of trace lengths, i.e., packet counts, in the datasets. All traces are truncated at 5,000 packets, as typical in related work for deep-learning classifiers [6, 50, 55]. The original Wang dataset is included for comparison. For one, note how splitting strategies result in a significant fraction of tiny traces, for the 25th percentile top-to-bottom in Figure 4: 57, 51, 68, and 155 packets for real-world, 73, 130, and 35 packets for closed simulated, 86, 153, and 52 packets for open simulated, and 496 packets for Wang k-NN [62].

While simulated datasets show a clear relationship between trace length and splitting strategy (mainly, number of paths), the real-world datasets are more messy. For all split datasets, we have less than 155 packets[2] for 25% of the traces. This is very little data in comparison to typical WF datasets—like Wang's, with a median of 1362 packets—highlighting why classifiers may struggle to become highly effective: what cannot be observed cannot be classified.

4 IMPROVED DEEP LEARNING ATTACKS

To properly evaluate defenses, it is important to investigate and optimize tailored attacks for each defense and dataset. In this section, we present *Maturesc*[3], a new state of the art attack with specific tailored sets of hyperparameters on a per-database basis that outperforms reported results of former attacks significantly, especially if only little training data is available.

Maturesc uses a modified version of optimized xresnets [22] which were configured for one-dimensional multi-channel input data and one-cycle-training [56]—to enable fast training with reduced amounts of epochs—and reduce overfitting when the amount of training data is limited. Maturesc is built in PyTorch [46] and fastai [28], available as open-source at https://github.com/m-bec/Splitting-Hairs-and-Network-Traces.

Earlier WF-attacks typically optimize hyperparameters once and then use a static set of hyperparameters for all datasets [6, 55]. We found that the hyperparameter tuning itself needs to be part of the attack for optimal results. This is similar to how Rimmer et al. [53] note the importance of hyperparameter tuning for evaluating traffic correlation attacks on different datasets. Our architectural decisions are based on optimizing learning efficiency when extensive hyperparameter tuning is part of the learning process.

We decided to limit ourselves to trace lengths of <5000 to be comparable to former attacks. Small improvements are possible if traces are particular long with a larger trace lengths. We use two input channels, i.e., timestamps and directional packet size. Both can be easily represented as separate input streams of fp32 numbers which are directly feed to our one-dimensional, two channel xresnets. This is highly efficient since no calculation intensive transformations are needed that way.

One-cycle-training enables learning in one go without the need for iterative fine tuning as long as fitting hyperparameters are chosen. We tested different architectural configurations and learning strategies but in our tests nothing achieved improved results compared to our chosen approach when combined with our systematic hyperparamter tuning.

For a more detailed insight into our architecture please see the following sections and the source code at https://github.com/m-bec/Splitting-Hairs-and-Network-Traces.

[2]While we use packets and cells interchangeably throughout, please do note that it is only CoMPS real-world that contains packets: the rest are Tor cells. In practice, multiple cells will be packed into a packet if possible, leaving even less data available by assumption.
[3]from Latin, Maturescere - approaching maturity.

```
$w \leftarrow$ Bernoulli( $\leftarrow\$ [0,1]$)
$n \leftarrow 0, b \leftarrow 0$
foreach packet do
    $n \leftarrow n+1$
    if $n > b$ then
        $p \leftarrow\$ w$
        $b \leftarrow$ Geometric$(1/20)$
        $n \leftarrow 0$
    endif
    send packet on path $p$
endforeach
```

(a) HyWF using two paths [23]

```
$w \leftarrow$ Dirichlet$(5)$
$n \leftarrow 0, b \leftarrow 0$
foreach packet do
    $n \leftarrow n+1$
    if $n > b$ then
        $p \leftarrow\$ w$
        $b \leftarrow\$ [50,70]$
        $n \leftarrow 0$
    endif
    send packet on path $p$
endforeach
```

(b) TrafficSliver's Batched Weighted Random (BWR) strategy using five paths [36]

```
$w \leftarrow$ Dirichlet$(3)$
$t \leftarrow 0$
foreach packet do
    if Since$(t) \geq$ 100ms then
        $p \leftarrow\$ w$
        $t \leftarrow$ Now$()$
    endif
    send packet on path $p$
endforeach
```

(c) CoMPS with Weighted Random (WR) strategy using three paths [60]

Figure 3: The most effective traffic splitting strategies of HyWF [23], TrafficSliver [36], and CoMPS [60]. Notation adjusted for ease of comparison, where w is the sampled weights assigned to each path, b the batch size of packets to send before updating, and p the sampled network path to send traffic over. Bernoulli, Dirichlet, and Geometric are sampled probability distributions with their respective arguments while $\leftarrow\$$ denotes uniformly random sampling in the specified range or probability vector. For time, Now() returns the current timestamp and Since(t) returns the duration of time since timestamp t.

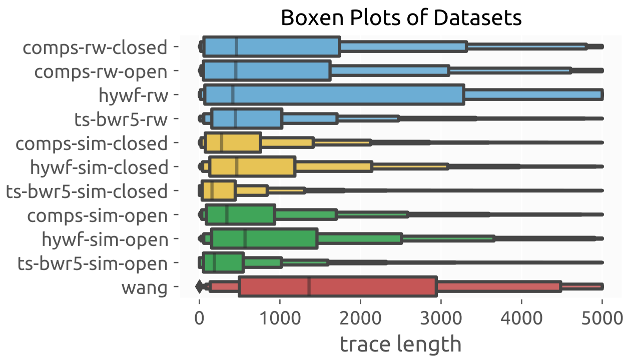

Figure 4: Distribution of trace lengths in datasets.

4.1 Setup

Before the training of the DL model begins, the data has to be brought in a unified format. In the WF community, this is typically done by representing traces using directional sequences or directional time (see Section 2.2). In our setup, we observed a small increase in accuracy when this is changed to relative timestamps instead: meaning that we are not counting time from the beginning of the trace but relative to the previous packet in the trace. We believe that a reason for that is the higher resistance against random delays in data transmission what makes learning a bit easier.

If available (only CoMPS), we also use the packet size as an additional input channel to our models. However, this also increases the memory footprint of the model by factor two.

For easy access to modern DL architectures and general improvements in the rapidly evolving area, we use PyTorch [46] and fastai [28]. In particular, the easy access to enhanced ResNet building-blocks that we utilize to build our xresnets [22], a efficient implementation of one-cycle training [56], and pre-built evaluation functions makes this a convenient choice.

For the hyperparameter tuning, we included all relevant ones in our attack script as command line parameters to enable an easy variation. The script is generated out of a Jupyter Notebook which was also used for testing and optimising the script.

The xresnets are configured for one dimensional multi channel data. Most successful were the configurations with 18 and 50 layers. The xresnets with only 18 layers excelled in two of the simulated open datasets (comps-sim-open and hywf-sim-open) were the batch size was also the largest with 2048.

We evaluated our attack with two NVIDIA A40 GPUs, an AMD EPYC 7713P 64-Core CPU, and 512 GB of memory. When utilizing a single GPU, training times were around two minutes per epoch for the real-world datasets and up to 33 minutes for the simulated TS-BWR5 open-world dataset on our 50 layer net, each with a trace length of 4900 packets/cells.

4.2 Journey

Here we describe some learning lessons from our training. Note that all datasets were split 8:1:1 for training, validation, and testing. We experimented with 10-fold cross-validation but noticed little to no variance, likely due to significant dataset sizes (see Table 1). This is similar to how our validation scores are close to our test scores (see shortly).

4.2.1 One-cycle Training. While providing the best results in general, One-cycle training has a stochastic component. Running another training session with the same hyperparameters might result in different outcomes. However, One-cycle training is considerably faster converging which allows to grind out near optimal results.

The speed-up that One-cycle-training provides can also be invested in a systematic approach to hyperparameter optimization. Such an extensive testing with different hyperparameter combinations is feasible since training times, e.g., for the real-world datasets are comparatively short.

A potential pitfall could be overfitting to the validation set, by choosing the training run with the best validation score. However, as we could validate, optimizations based on a validation set translate very well to test sets.

4.2.2 Hyperparameter Tuning. To enable its full potential, One-cycle training needs different hyperparameters compared to classical fitting to work properly. In general our chosen learning rates are higher (up to 0.01) compared to other work since they reflect maximum learning rates that are only reached around the first quarter of the training. For the real-world datasets we are also using a relatively small batch size of 32 to 128 items.

However, for the much bigger simulated datasets, especially the open world ones, we realised quickly that similar batch sizes lead to no learning at all or to overfitting. Here, bigger batch sizes (350 to 2048), and in addition, strong regulation (dropout of 85%) are necessary to receive the best results.

We trained on all datasets with multiple permutations of hyperparameters. For the batch size we tested 32, 64, 128, 256, and depending on the GPU memory consumption 350 or 400, 512, 1024, 2048. For the learning rate we tested 0.002, 0.001, 0.05, 0.02, 0.01. For dropout we trained with 0%, 50%, 75%, 85%, and 90%. For the amount of epochs we trained with 10, 20, 30, and 40. The amount of layers and the amount of entries per trace that were used was determined by available memory on the GPU since more is generally better here but the trade-off with batch size has to be considered.

4.3 Results

The optimal hyperparamters were chosen based on the accuracy on the validation set. The best combination of hyperparameters for each dataset can be seen in Table 2.

We realized that in all our tests the test set scores are relatively close to the validation scores with an average absolute distance of 1.21% accuracy (+/- 0.19% with α = 0.005) and a maximum of less than 3.9% absolute distance. In all but two cases, the best models based on validation scores were also the best models based on test scores.

It is important to recall that DL results are stochastic by nature. Repeating an experiment with the same hyperparameters on the same hardware can produce slightly different results. Another factor are trade-offs between parameters. Especially learning rate, batch size, and dropout cross-influence each other. So can for example an increase of the batch size while also increasing the learning rate result in very similar results. Often even moderate variations can result in similar results. To give an example, comps-rw-closed gave the following results with a batch size of 64, 128, and 256 which resulted in 75.8%, 75.3%, and 73.3% validation set accuracy and 76.0%, 75.0%, and 73.2% test set accuracy respectively.

Figure 5 shows our results, including the state-of-the-art WF attacks DF [55], Tik-Tok [50], and Var-CNN [6] for reference. Closed-world results are plotted as accuracy–confidence threshold graphs, particularly useful in our setting to show when the Var-CNN ensemble loses confidence in its classification. Open-world results are shown as as precision–recall graphs, demonstrating the inherent trade-off between prediction value and sensitivity.

We start directly comparing our results with those reported in the work of respective splitting strategy:

- For HyWF [23], Henri et al. report two relevant results. On the same real-world dataset they state 49.2% accuracy using DF, compared to our 66.7% accuracy. By repeatedly simulating HyWF on the k-NN Wang et al. dataset [62]—identical to our setting—they reach slightly below 50% accuracy, compared to our 71.6% accuracy.
- For CoMPS [60], Wang et al. report for their real-world dataset a F_1 score of 18.3% with DF and 32.9% with Var-CNN, compared to our 72.4%.
- For TS BWR5 [36], De la Cadena et al. report 8.1% accuracy for DF on their real-world dataset, compared to our 53.8% accuracy.

Going over the real-world dataset results (first row of Figure 5) we see a significant gain in classifier performance over state-of-the-art. In terms of closed-world results for CoMPS on the same dataset (not shown in figure), we get 76.0% accuracy, compared to second-best Var-CNN at 50.7%. The difference between our results for TS BWR5 (53.8%) and CoMPS (76.0%) can be explained in part by the reduction in paths, making more data available. HyWF is an outlier here, with only two paths but only 66.7% accuracy: this can likely be explained by the low utility of time in the HyWF real-world dataset. This is apparent from the poor accuracy of Tik-Tok (4.2%), which is consistent with the directional time classifier part of the Var-CNN ensemble also performing poorly (4.4%, not shown in figure, but note the drop in accuracy once the confidence threshold goes past 0.5). We suspect this is an artifact of how the dataset was collected, time is still very useful for classifiers against HyWF, see next.

Looking at closed-world performance on the simulated datasets (second row of Figure 5), we see a much cleaner relationship between number of paths and accuracy: 71.6% for HyWF (two paths), 59.0% for CoMPS (three paths), and 43.5% for TS BWR5 (five paths). Given the huge availability of traces (see Table 1) to train on, Tik-Tok is the clear second-best with the second-highest accuracy as well as smooth confidence threshold. This is a compelling result since Tik-Tok uses a simpler architecture (DF) and is faster to train than Var-CNN. Note that Var-CNN was designed to be more efficient with less training data, so the comparison in training time and setting (massive amount of training data) is not fair.

Finally, the open-world results on the simulated datasets (third row of Figure 5) show similar results as in the closed-world. The Var-CNN ensemble behaviour of drastically getting worse somewhere in the middle of the threshold is also visible here. Tik-Tok remains the clear second-best.

Interestingly, our Maturesc classifier never reaches close to 100% precision, even with a > 0.99 threshold: we get HyWF 94.9%, CoMPS 90.6%, and TS-BWR5 93.1% precision. However we use the fastai CrossEntropyLossFlat loss function which flattens input and output. This is different from the loss functions used by the other attacks. In the end, our model seems to be more hesitant to give absolute certainties about specific results what is arguably more realistic.

Table 2: Optimized hyperparameters for each dataset based on validation set accuracy (vali acc), with trace length = maximum amount of entries per trace that are used; layers = layers of the used architecture; learning rates reflects maximum learning rate; vali acc = rounded validation set accuracy; test acc = rounded test set accuracy.

dataset	trace length	layers	dropout	batch size	learning rate	epochs	vali acc	test acc
comps-rw-closed	4900	50	50%	64	2e-2	30	75.8%	76.0%
comps-rw-open	4900	50	50%	64	2e-2	30	77.5%	78.5%
hywf-rw	4900	50	0%	32	2e-2	30	69.2%	66.7%
ts-bwr5-rw	4900	50	50%	128	2e-2	20	54.2%	53.8%
comps-sim-closed	4900	50	85%	350	2e-2	10	62.6%	59.0%
hywf-sim-closed	4900	50	85%	350	1e-2	10	74.8%	71.6%
ts-bwr5-sim-closed	4900	50	85%	512	2e-2	20	46.6%	43.5%
comps-sim-open	2500	18	85%	2048	2e-2	30	75.1%	72.7%
hywf-sim-open	4900	18	85%	2048	2e-2	30	82.5%	80.5%
ts-bwr5-sim-open	4900	50	85%	512	2e-3	30	69.1%	67.7%

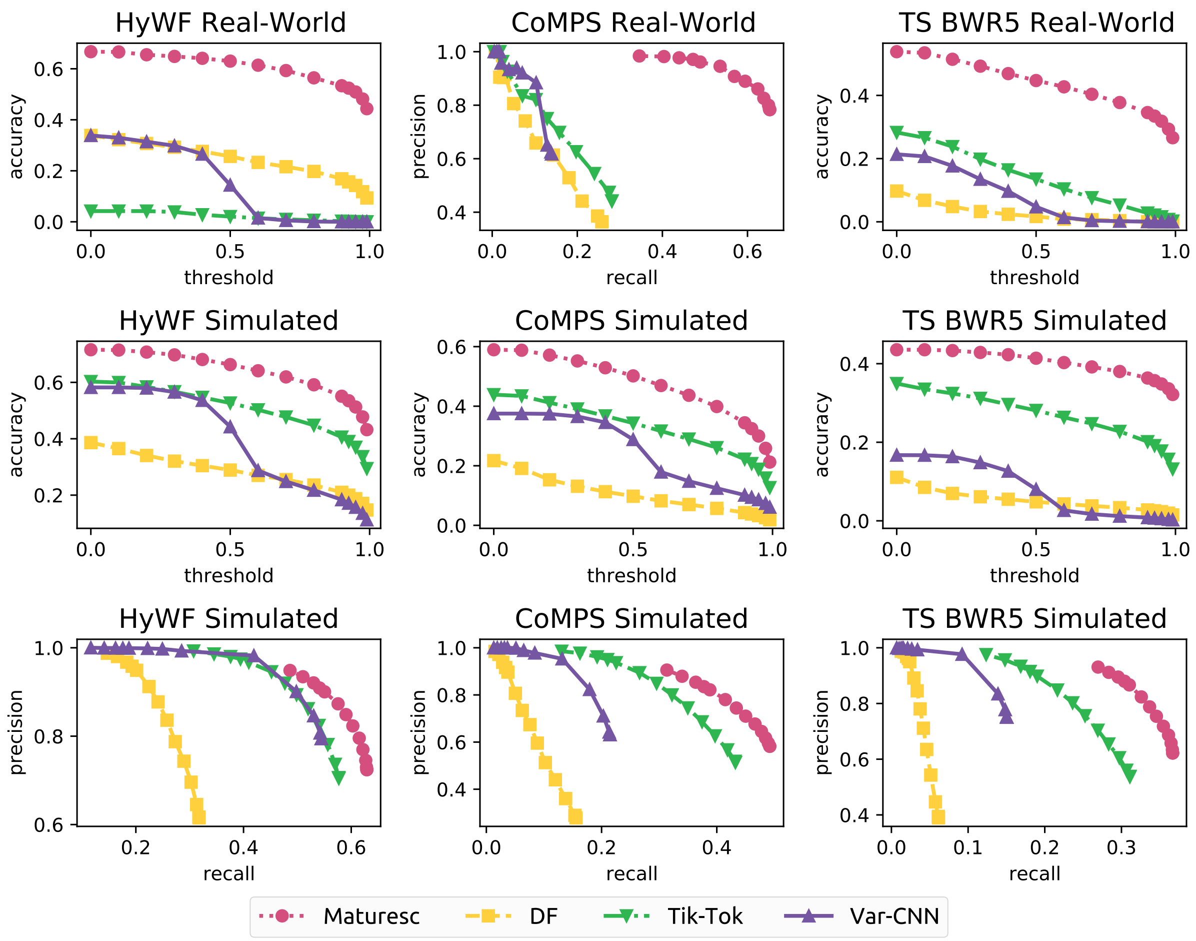

Figure 5: Evaluation results of our Maturesc classifier compared to Deep Fingerprinting (DF) [55], Tik-Tok [50], and Var-CNN [6]. The first row contains results for the real-world datasets of HyWF [23], CoMPS [60], and TrafficSliver BWR5 [36], while the second and third rows show simulated datasets based on the widely used Wang k-NN dataset [62].

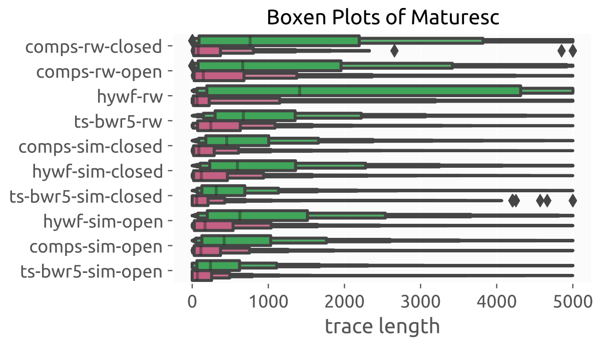

Figure 6: Trace lengths of correct (green, above) and wrong (red, below) classifications by the Maturesc classifier for each dataset. Wrongly classified traces are significantly shorter.

5 DISCUSSION

We discuss lessons learned and implications of our results, ranging from looking closer at the DL attacks to the setting of traffic splitting as a WF defense.

5.1 Finding Optimal Hyperparameters

Training optimal models highly depends on the type and amount of training data. We found some general rules of thumb:

- The bigger the dataset, the bigger the batch size should be (but still not too big) and the more regulation is helpful, i. e., dropout.
- Bigger models (i.e. more layers) increase performance but not by much.
- More than 30 epochs of training were not helpful.

5.2 What is Wrongly Classified?

Figure 6 shows boxen blots for each dataset with the trace lengths of correctly and wrongly classified traces using our improved classifier from Section 4 (same data as plotted in Figure 5). From top to bottom in Figure 6, the median trace lengths for wrongly classified are 79, 146, 64, 244, 87, 120, 60, 169, 115, and 68 packets. Recall (Section 3) that 25% of traces in all datasets have less than 155 packets in them, accounting for 25% of accuracy in a closed-world dataset. It is apparent that the classifier *fails mainly on very short traces*. This failure is *across all websites/classes*: every dataset but one has at least one sample from *each class* wrongly classified (except for `hywf-rw`, where 99/100 classes were wrongly classified).

One of the conclusions of De la Cadena et al. [36] when analyzing TS BWR5 was that good protection against WF attacks were achieved if *"less than 60% of the total length"* was available to the attacker. Based on our results and improved attacks, we believe a more accurate assessment is expressed in terms of an absolute trace length in the order of hundreds of packets. Once the attacker is able to observe a couple of hundred of packets, especially for closed-world evaluations, the protection drops drastically. This is more likely to occur the fewer paths traffic is split over.

5.3 On Traffic Splitting as a Defense

We believe that our results warrant a more critical view on traffic splitting as a WF defense. In a sense—given that attacks are in general successful if the attacker can observe a couple of hundred packets of a trace—traffic splitting is more of an assumption than a defense. The underlying assumption is that the attacker cannot observe the vast majority of traffic. This leads us to conclude that—while it is possible to construct a WF defense using a mostly unobservable communication path—it is a very strong assumption in the context of anonymous communication [48].

For one, using splitting as a WF defense, we do not know which path is unobservable to the attacker. As in all considered traffic splitting strategies, one can just randomly bias the path selection (see each first line in Figure 3). Our results imply that this bias should be very high for one path: that way either traffic is unobservable and safe, or it is not. Spreading traffic over multiple paths just greatly increases the chances that a WF attacker can observe enough traffic to attack.

Noteworthy, the heavy single-path bias is similar to what Henri et al. [23] observed when evaluating the default MPTCP scheduler ("minRTT") as a splitting defense. They noted that because minRTT aims to minimize RTT, it typically ends up sending almost all traffic over the path with the lowest RTT ("primary path"), resulting in the attacker having only a 2% success rate on classifying traffic on the other ("secondary") path (so, at most, 52% accuracy if the primary path is perfectly classified). As a splitting defense, minRTT would actually perform just as well as the best splitting strategy evaluated here (TS BWR5 had the lowest real-world accuracy of 53.8% accuracy, see Section 4) using only two paths, compared to the five of TS BWR5. Using five paths with minRRT would likely result in ≈20–30% accuracy, under the assumption that minRRT would still mostly end up favoring the fastest path. This is in a sense a *straw man traffic splitting strategy* that uniformly random selects one available path and sends all traffic over it, resulting in an expected $1/N$ accuracy for N paths. One can think of this as a baseline for any splitting strategy to compare against.

Efficiently using multiple paths, especially if paths have varying latency and/or throughput, is not trivial [16, 31, 40]. In asymmetric latency settings, schedulers like BLEST [16] will emphasize using the faster path to an even greater extent than minRTT for web traffic (e.g., see Figure 9 in the paper of BLEST [16]). It is first for symmetric paths or for other tasks than web traffic, like bulk downloads, where more than one path will end up seeing notable use. As a consequence—in asymmetric latency settings of web-traffic—currently proposed traffic splitting defenses will result in more overhead than expected in practise: by using non-optimal secondary paths performance will get worse. How much worse depends on how bad some of the bad paths are. For Tor, slow paths are likely to include two continental hops of added latency (e.g., circuits from EU → US → EU or Asia → US → EU), potentially to destinations a third continental hop away.

The Tor Project is working on proposal 329 [18] for traffic splitting. The proposal suggests splitting over *two* circuits (paths) to a single exit, based on Conflux [4] (see related work in Section 6), with scheduling/splitting algorithms based on minRTT and BLEST to be investigated. It further notes that splitting may help against WF attacks, citing the work of De la Cadena et al. on TrafficSliver [36]. Our improved results on TS BWR5 shows that minRTT or BLEST would both make better WF defenses on their own by simply being likely to send the vast majority of traffic over the primary circuit and hoping that it is unobservable by the attacker.

Since minRTT and BLEST-like algorithms will heavily favour the path with the lowest latency, traffic splitting is closely related to path selection. In a sense, traffic splitting becomes an extension of path selection, *after* paths have been selected. Consider the following two scenarios:

- If paths of similar latency are selected, then the splitting algorithm may end up sending traffic over all paths. From a WF perspective, this is undesirable, because then attackers along both paths can launch successful attacks.
- If the selected paths have different latency characteristics, then the splitting algorithm will rapidly detect this fact and only send traffic over the primary path.

The above suggests that the path selection algorithm should try to provide paths of similar latency characteristics as per its own estimates (more generally, use similar metrics as the traffic splitting algorithm), otherwise it is very likely that including the second path is mostly pointless. In this scenario, additional WF defenses are needed. Our results show that with a few hundreds of packets/cells observed traffic splitting offers little to no defense on its own. As was proposed by Henri et al. [23] and Wang et al. [60], splitting could be combined and evaluated in conjunction with other WF defenses. However, for as long as path selection and splitting algorithms are likely to send virtually all traffic over a single path in some cases, WF defenses may as well be evaluated without considering traffic splitting at all.

5.4 Assuming Unobservable Paths in Tor

In general, one cannot know which path is unobservable to a passive adversary. For Tor, clients would pick at least the number of guards as the number of paths used for traffic splitting (in line with proposal 329 [18]). At the time of writing, the top-three countries by aggregated guard probability are Germany (39%), US (11%), and France (10%). By virtue of the guard probability, the vast majority of users would end up picking guards outside of the country they are connecting from. Nation state actors presumably observe such traffic crossing their national borders and trivially link paths by source IP-address, invalidating the unobservability assumption for this powerful and highly relevant threat actor [11].

ISPs of users, unless users do multihoming (presumably rare today, and will likely remain so for some time for mobile-first users), can also observe all paths and trivially link them. The same is true for WiFi-hotspots, mobile carriers, and more broadly the AS the user is connecting from. For WF, the only entities negatively impacted by traffic splitting are non-colluding non-overlapping ASes on the paths between a clients connecting AS and each guard, and the guards.

Arguably, the biggest win of traffic splitting is limiting the observability of paths by attackers operating as guards, because it is straightforward to operate one for anyone with modest resources. Because guard sets are kept by clients for months, traffic splitting would have the benefit of only letting a malicious guard observe a fraction of clients traffic (expected proportional to the size of the guard set). Guards are also in a position to see more fine-grained cell-level events in a circuit, while other network-level attackers have to deal with multiplexed TLS between the client and guard that may somewhat reduce the effectiveness of attacks [34, 44, 63].

6 RELATED WORK

Traffic splitting in Tor has its origins in the Conflux scheme by AlSabah et al. [4]. Conflux proposed splitting traffic over two network paths—built from the client to different guards and merged at the exit. The goal was to improve performance by splitting traffic proportional to the measured latency of each path. Conflux is stated as the primary inspiration for proposal 329 by the Tor Project [18]. Earlier, Snader [57] showed that multiple circuits can improve performance, highlighted by Conflux as important related work. Our work only focuses on traffic splitting as a WF defense.

Traffic splitting as a WF defense for Tor was first proposed in 2019 [37] by De la Cadena et al. in poster-format followed by detailed investigations in 2020 by De la Cadena et al. with TrafficSliver BWR [36] and Henri et al. with HyWF [23]. Traffic splitting was then in turn later generalized and expanded by Wang et al. [60] with the CoMPS framework for any service supporting connection migration. In addition to proposing the splitting strategies analyzed in this paper, De la Cadena et al., Henri et al., and Wang et al. investigated a number of other potential splitting strategies, e.g., round-robin, fixed splitting probabilities, splitting by direction, and weighted random. The splitting strategies were evaluated using primarily k-fingerprinting [20] and DF [55] (Wang et al. in addition used Var-CNN) and rejected due to not significantly reducing accuracy. This is similar to our methodology of focusing on WF classifier accuracy.

Henri et al. and Wang et al. both propose to combine their traffic splitting strategies with other WF defenses, in particular WTF-PAD [35] and Walkie-Talkie [64]. All of their evaluations show a significant reduction in attacker accuracy. Our results confirm that combining traffic splitting with other WF defenses is needed.

De la Cadena et al. further present TrafficSliver-App, an application layer defense that (when using its most successful splitting strategy) sends each HTTP request over one of *seven* random circuits and achieves 57.34% accuracy against DF. We did not evaluate TrafficSliver-App, but note that the accuracy is well above the straw-man design of 1/7 accuracy. TrafficSliver-App only requires client-side changes though—making it easier to implement—so it may be interesting to consider as part of WF defenses.

More broadly, path selection is essential for providing anonymity in anonymity networks, especially for low-latency anonymity networks like Tor [14]. For mix networks, Serjantov and Murdoch already in 2005 considered splitting messages across network paths [54]. For Tor, path selection can be optimized in different ways, such as to take into account AS-level attackers [13], network congestion [61], and optimizing for low-latency [3]. In general, optimizing for any performance metric comes at a cost of anonymity, involving trade-offs [33, 59]. Our work relates to what takes place after path selection is done, and shows that if splitting/scheduling algorithms use latency as a performance metric, then likely path selection should as well.

Juarez et al. [34] and Perry [47] critically evaluate simplified assumptions surrounding the evaluation of WF attacks, e.g., closed-vs-open world, front-page vs web-page fingerprinting, base-rate concerns, and the absence of background traffic in collected network traces, leading to a series of works investing some of the concerns (e.g., [44, 49, 63]). In a sense, our work is a dual of these works, where we focus on the key underlying assumption of unobservable network paths made by a category of defenses rather than attacks.

7 CONCLUSIONS

In this paper we investigated the effectiveness of different types of traffic splitting as a defense against improved Website Fingerprinting (WF) attacks. We showed that our improved attack, Maturesc, works effectively even when only a few hundreds cells/packets are observable by the attacker. Further advances in Deep Learning are expected to enable attacks that need even less observable data. Given that so little data is needed for successful attacks, the risk of sending traffic over multiple (potentially compromised) paths when splitting may outweigh any potential gains. At least for all splitting strategies we evaluated, none offered more protection than the straw man defense of just uniformly randomly picking a path and sending all traffic over it (see Sections 4 and 5.3). For HyWF and its setting of two paths, the straw-man defense would at most see 50% accuracy, Maturesc achieved 66.7%. For CoMPS with three paths, the straw-man defense would achieve 33% accuracy, Maturesec got 76.0%. Finally, for TS BWR5 with five paths, the straw-man defense would have an accuracy of 20%, Maturesc achieved 53.8%. Clearly, on average, the straw-man defense would offer superior protection against a WF attacker in all those settings.

The weak threat model underpinning traffic splitting is that all but one path used for communication is unobservable. This is a strong assumption for anonymous communication [48]. Many relevant threat actors, such as ISPs (of clients lacking multihoming) and presumably nation state actors are likely to observe more than one path or even all paths. Even ignoring the aforementioned arguments, traffic splitting would not work as a defense in these cases. It would be anyway necessary to deploy additional defenses against such realistic, more powerful attackers that are squarely within the threat model of Tor [11]. In fact, this is the traditional (non traffic-splitting) setting of WF.

Since, in general, the considered traffic splitting strategies cannot protect observable paths, additional WF defenses are needed. We conclude that traffic splitting should primarily be seen as a technique to increase performance, not a technique for defending against WF attacks.

ARTIFACT AVAILABILITY

Datasets, documentation, and Python scripts for the Maturesc attack and splitting simulations are available at:

https://github.com/m-bec/Splitting-Hairs-and-Network-Traces

ETHICAL CONSIDERATIONS

We believe that it is both ethical and in the interest of society to make our improved attacks public, if anything because already public WF attacks (like DF [55] and Var-CNN [6]) are comparable on *undefended* traffic, i.e., the state of Tor and less defended protocols like WireGuard today. Our research is also focused on proposed defenses: by improving the evaluation of defenses before they are deployed we hope to contribute to the effective deployment of WF defenses in the future.

ACKNOWLEDGEMENTS

The authors would like to thank Per Hurtig and Mike Perry for helpful comments and discussion. This work has been funded in part by Mullvad VPN and the Swedish Internet Foundation.

REFERENCES

[1] Kota Abe and Shigeki Goto. 2016. Fingerprinting attack on Tor anonymity using deep learning. *Proceedings of the Asia-Pacific Advanced Network* 42 (2016), 15–20.

[2] Daniel Agnew. 2020. Google Trends Reveals Surge in Demand for VPN. https://www.namecheap.com/blog/vpn-surge-in-demand/.

[3] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. 2012. LASTor: A Low-Latency AS-Aware Tor Client. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 476–490. https://doi.org/10.1109/SP.2012.35

[4] Mashael AlSabah, Kevin Bauer, Tariq Elahi, and Ian Goldberg. 2013. The path less travelled: Overcoming Tor's bottlenecks with traffic splitting. In *PETS*.

[5] Apple. 2021. iCloud Private Relay Overview. https://www.apple.com/privacy/docs/iCloud_Private_Relay_Overview_Dec2021.PDF.

[6] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2019. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proc. Priv. Enhancing Technol.* 2019, 4 (2019), 292–310. https://doi.org/10.2478/popets-2019-0070

[7] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *ACM SIGSAC*. 227–238. https://doi.org/10.1145/2660267.2660362

[8] Heyning Cheng and Ron Avnur. 1998. Traffic analysis of SSL encrypted web browsing. *Project paper, University of Berkeley* (1998).

[9] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2018. Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency - Choose Two. In *IEEE SP*. 108–126. https://doi.org/10.1109/SP.2018.00011

[10] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2020. Comprehensive Anonymity Trilemma: User Coordination is not enough. *Proc. Priv. Enhancing Technol.* 2020, 3 (2020), 356–383. https://doi.org/10.2478/popets-2020-0056

[11] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security*.

[12] Jason A. Donenfeld. 2017. WireGuard: Next Generation Kernel Network Tunnel. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society.

[13] Matthew Edman and Paul F. Syverson. 2009. As-awareness in Tor path selection. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis (Eds.). ACM, 380–389. https://doi.org/10.1145/1653662.1653708

[14] Nick Feamster and Roger Dingledine. 2004. Location diversity in anonymity networks. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004*, Vijay Atluri, Paul F. Syverson, and Sabrina De Capitani di Vimercati (Eds.). ACM, 66–76. https://doi.org/10.1145/1029179.1029199

[15] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS Adoption on the Web. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, Engin Kirda and Thomas Ristenpart (Eds.). USENIX Association, 1323–1338. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/felt

[16] Simone Ferlin, Özgü Alay, Olivier Mehani, and Roksana Boreli. 2016. BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks. In *2016 IFIP Networking Conference, Networking 2016 and Workshops, Vienna, Austria, May 17-19, 2016*. IEEE Computer Society, 431–439. https://doi.org/10.1109/IFIPNetworking.2016.7497206

[17] Jiajun Gong and Tao Wang. 2020. Zero-delay Lightweight Defenses against Website Fingerprinting. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 717–734. https://www.usenix.org/conference/usenixsecurity20/presentation/gong

[18] David Goulet and Mike Perry. 2020. Overcoming Tor's Bottlenecks with Traffic Splitting. /https://gitlab.torproject.org/tpo/core/torspec/-/raw/main/proposals/329-traffic-splitting.txt.
[19] Jamie Hayes and George Danezis. 2015. Guard Sets for Onion Routing. *PETS* (2015).
[20] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 1187–1203. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes
[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.
[22] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. 2019. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 558–567.
[23] Sébastien Henri, Gines Garcia-Aviles, Pablo Serrano, Albert Banchs, and Patrick Thiran. 2020. Protecting against Website Fingerprinting with Multihoming. *PETS* (2020). https://doi.org/10.2478/popets-2020-0019
[24] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *CCSW.*
[25] Andrew Hintz. 2002. Fingerprinting Websites Using Traffic Analysis. In *PET.*
[26] Paul E. Hoffman and Patrick McManus. 2018. DNS Queries over HTTPS (DoH). RFC 8484. https://doi.org/10.17487/RFC8484
[27] James K Holland and Nicholas Hopper. 2022. RegulaTor: A Straightforward Website Fingerprinting Defense. *PETS* (2022).
[28] Jeremy Howard and Sylvain Gugger. 2020. Fastai: a layered API for deep learning. *Information* 11, 2 (2020), 108.
[29] Zi Hu, Liang Zhu, John Heidemann, Allison Mankin, Duane Wessels, and Paul E. Hoffman. 2016. Specification for DNS over Transport Layer Security (TLS). RFC 7858. https://doi.org/10.17487/RFC7858
[30] Christian Huitema, Sara Dickinson, and Allison Mankin. 2022. DNS over Dedicated QUIC Connections. RFC 9250. https://doi.org/10.17487/RFC9250
[31] Per Hurtig, Karl-Johan Grinnemo, Anna Brunström, Simone Ferlin, Ozgu Alay, and Nicolas Kuhn. 2019. Low-Latency Scheduling in MPTCP. *IEEE/ACM Trans. Netw.* 27, 1 (2019), 302–315. https://doi.org/10.1109/TNET.2018.2884791
[32] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000. https://doi.org/10.17487/RFC9000
[33] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul F. Syverson. 2013. Users get routed: traffic correlation on tor by realistic adversaries. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM, 337–348. https://doi.org/10.1145/2508859.2516651
[34] Marc Juárez, Sadia Afroz, Gunes Acar, Claudia Díaz, and Rachel Greenstadt. 2014. 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*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM, 263–274. https://doi.org/10.1145/2660267.2660368
[35] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Díaz, and Matthew Wright. 2016. Toward an Efficient Website Fingerprinting Defense. In *ESORICS.* 27–46. https://doi.org/10.1007/978-3-319-45744-4_2
[36] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. 2020. TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting. In *CCS.*
[37] Wladimir De la Cadena, Asya Mitseva, Jan Pennekamp, Jens Hiller, Fabian Lanze, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. 2019. POSTER: Traffic Splitting to Counter Website Fingerprinting. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 2533–2535. https://doi.org/10.1145/3319535.3363249
[38] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[39] Marc Liberatore and Brian Neil Levine. 2006. Inferring the source of encrypted HTTP connections. In *CCS.*
[40] Yeon-sup Lim, Erich M. Nahum, Don Towsley, and Richard J. Gibbens. 2017. ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2017, Incheon, Republic of Korea, December 12 - 15, 2017.* ACM, 147–159. https://doi.org/10.1145/3143361.3143376
[41] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. 2011. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011.* The Internet Society. https://www.ndss-symposium.org/ndss2011/httpos-sealing-information-leaks-with-browser-side-obfuscation-of-encrypted-flows
[42] Jonathan Magnusson. 2021. *Evaluation of a Proposed Traffic-Splitting Defence for Tor: Using Directional Time and Simulation Against TrafficSliver.* Master's thesis. Karlstad University, Department of Mathematics and Computer Science.
[43] Rishab Nithyanand, Xiang Cai, and Rob Johnson. 2014. Glove: A Bespoke Website Fingerprinting Defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society, WPES 2014, Scottsdale, AZ, USA, November 3, 2014*, Gail-Joon Ahn and Anupam Datta (Eds.). ACM, 131–134. https://doi.org/10.1145/2665943.2665950
[44] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale. In *NDSS.*
[45] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. 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*, Yan Chen and Jaideep Vaidya (Eds.). ACM, 103–114. https://doi.org/10.1145/2046556.2046570
[46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
[47] Mike Perry. 2013. A Critique of Website Traffic Fingerprinting Attacks. https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks.
[48] Andreas Pfitzmann and Marit Hansen. 2010. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management.
[49] Tobias Pulls and Rasmus Dahlberg. 2020. Website Fingerprinting with Website Oracles. *PETS* (2020).
[50] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks. *Proc. Priv. Enhancing Technol.* 2020, 3 (2020), 5–24. https://doi.org/10.2478/popets-2020-0043
[51] Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi. 2022. VPNalyzer: Systematic Investigation of the VPN Ecosystem. In *Network and Distributed System Security.*
[52] Vera Rimmer, Davy Preuveneers, Marc Juárez, Tom van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *NDSS.* http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_03A-1_Rimmer_paper.pdf
[53] Vera Rimmer, Theodor Schnitzler, Tom van Goethem, Abel Rodríguez Romero, Wouter Joosen, and Katharina Kohls. 2022. Trace Oddity: Methodologies for Data-Driven Traffic Analysis on Tor. *Proc. Priv. Enhancing Technol.* 2022, 3 (2022), 314–335. https://doi.org/10.56553/popets-2022-0074
[54] Andrei Serjantov and Steven J. Murdoch. 2005. Message Splitting Against the Partial Adversary. In *Privacy Enhancing Technologies, 5th International Workshop, PET 2005, Cavtat, Croatia, May 30-June 1, 2005, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 3856)*, George Danezis and David M. Martin Jr. (Eds.). Springer, 26–39. https://doi.org/10.1007/11767831_3
[55] Payap Sirinam, Mohsen Imani, Marc Juárez, and Matthew Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *CCS.*
[56] Leslie N Smith. 2018. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820* (2018).
[57] Robin A Snader. 2009. *Path selection for performance-and security-improved onion routing.* University of Illinois at Urbana-Champaign.
[58] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. 2002. Statistical Identification of Encrypted Web Browsing Traffic. In *IEEE S&P.*
[59] Chris Wacek, Henry Tan, Kevin S. Bauer, and Micah Sherr. 2013. An Empirical Evaluation of Relay Selection in Tor. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013.* The Internet Society. https://www.ndss-symposium.org/ndss2013/empirical-evaluation-relay-selection-tor
[60] Mona Wang, Anunay Kulshrestha, Liang Wang, and Prateek Mittal. 2022. Leveraging strategic connection migration-powered traffic splitting for privacy. *PETS* (2022).
[61] Tao Wang, Kevin S. Bauer, Clara Forero, and Ian Goldberg. 2012. Congestion-Aware Path Selection for Tor. In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 7397)*, Angelos D. Keromytis (Ed.). Springer, 98–113. https://doi.org/10.1007/978-3-642-32946-3_9
[62] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security.* 143–157. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/wang_tao

[63] Tao Wang and Ian Goldberg. 2016. On Realistically Attacking Tor with Website Fingerprinting. *PoPETs* 2016, 4 (2016), 21–36. https://doi.org/10.1515/popets-2016-0027

[64] Tao Wang and Ian Goldberg. 2017. 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*, Engin Kirda and Thomas Ristenpart (Eds.). USENIX Association, 1375–1390. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/wang-tao

A REGULAR WF EFFECTIVENESS

We briefly evaluated RegulaTor-heavy by Holland and Hopper [27] using Maturesc on the Wang et al. [62] dataset, for which they reported an accuracy of 17.8%. Maturesc achieved 24.1% with default parameters used for the real-world dataset. This comparatively small increase is a testament to how WF defenses not based on traffic splitting are more robust against modern Deep Learning attacks, while also demonstrating that Maturesc is not only useful against traffic splitting defenses.