Tor Traffic Analysis: Data-driven Attacks and Defenses

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

James Kevin Holland Jr.

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Nicholas Hopper

July, 2024

Acknowledgements

First, I'd like to thank my advisor, Nick Hopper. His extensive knowledge and expert guidance made this work possible.

I also thank the other committee members — Kangjie Lu, Chang Ge, and Andrew Odlyzko — for their time and support.

I've been lucky to have an array of skilled collaborators and co-authors, including Se Eun Oh, Taiji Yang, Ethan Witwer, Jason Carpenter, Nate Matthews, Mohammad Saidur Rahman, and Matthew Wright. Their thoughts and input shaped this work for the better, and I learned a great deal from them. In particular, I'd like to thank Se Eun Oh for helping me get up to speed with Tor research and developing much of the prior work for this dissertation. I also greatly appreciate Nate Matthews' work providing implementations for some of the traffic analysis techniques discussed in this dissertation

Furthermore, I thank everyone else I worked alongside in graduate school: Saugata Paul, Jaskaran Veer Singh, Devon Tuma, and Eric Brigham.

I am profoundly grateful to my family, my fiancée, and my friends for their unwavering encouragement and love.

Dedication

To my parents, who supported me all this way

Abstract

Anonymity networks such as Tor aim to protect the confidentiality of both the data and metadata — who communicates with whom — of their users. However, traffic analysis techniques can exploit the frequency and timing of network traffic to expose the metadata of these communications. These techniques include website fingerprinting and end-to-end flow correlation, both thoroughly discussed in previous literature. In a website fingerprinting (WF) attack, an adversary between the user and the first Tor relay records the timing and volume of Tor traffic to determine which website the user is visiting. In a flow correlation (FC) attack, the adversary records traffic metadata at both the entry and exit points of the Tor network and then attempts to correlate these flows, thus breaking Tor's anonymity.

Our first objective is to demonstrate that, despite challenges such as variable network conditions, the large sets of webpages associated with many websites, and potential "padding" to prevent traffic analysis, these attacks can be executed effectively. For instance, our Convolutional vision Transformer (CvT) approach, which merges the relative strengths of convolutional neural networks and transformers, can be used with multi-channel feature representations to significantly enhance attack accuracy against defended traffic. This demonstrates that website fingerprinting attacks can be successful even when traffic is obfuscated.

Another potential barrier for traffic analysis attacks against Tor is the arbitrarily large amount of non-targeted traffic that an attacker must differentiate from monitored web pages. In particular, an attacker interested in a small subset of traffic may find that the number of false positives begins to surpass the limited number of true positives. Accordingly, in the website fingerprinting setting, we present 'precision optimization' techniques to ensure that an attacker can identify a substantial subset of web pages with high precision.

Next, we present a series of performance improvements to current state-of-the-art flow correlation techniques, demonstrating that the improved techniques can operate with even higher accuracy and reliability. Furthermore, we apply these techniques to the problem of stepping-stone identification, showing that our approach can be adapted to correlate flow pairs that have been sent through multiple intermediate hosts.

Given the success of these traffic analysis techniques, we then develop defenses to minimize

the likelihood of successful website fingerprinting or flow correlation attacks. Although various defenses exist, most are either ineffective, introduce high latency and bandwidth overhead, or require additional infrastructure. Therefore, we aim to design defenses that are both effective and efficient. The first defense, RegulaTor, leverages common patterns in web browsing traffic to regularize traffic, significantly reducing the performance of website fingerprinting attacks while incurring moderate bandwidth and low latency overhead.

For latency-sensitive users, we propose DeTorrent, which uses competing neural networks to create and evaluate traffic analysis defenses that insert fake traffic into real traffic flows. DeTorrent operates with moderate overhead and defends against both website fingerprinting and flow correlation attacks more effectively than comparable padding-only defenses. We also demonstrate DeTorrent's practicality by deploying it alongside the Tor network, ensuring it maintains performance when applied to live traffic.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As people's personal and professional lives increasingly depend on the Internet, the threat of network surveillance has become particularly salient. As a result, the anonymity network Tor has surged in popularity, recording millions of users each day. Tor works by routing internet traffic through a network of volunteer-operated relays such that no relay knows both the origin and destination of the traffic. Tor uses multi-layered encryption, where each relay decrypts the outer layer to reveal the next destination, ensuring secure and anonymous communication.

However, it is inherently difficult to hide internet traffic timing information in a low-latency anonymity network. Accordingly, adversaries between a user and Tor may be able to carry out a website fingerprinting attack to determine the destination of web browsing traffic, compromising Tor's privacy. The same timing information and traffic metadata can also be used by an adversary on both the entry and exit side of Tor to correlate entry and exit flows, thus determining which users are visiting which websites.

The extent to which Tor is vulnerable to these attacks has been debated in prior research [1, 2, 3, 4]. However, recent attacks have achieved high accuracy, even in challenging and data-limited settings [5, 6, 7, 8]. To further demonstrate this, we first present an improved website fingerprinting attack that is highly effective, even against defended traffic. Because achieving high precision in a realistic setting has been an area of concern for website fingerprinting attacks, we then present multiple precision optimization techniques to ensure that WF attacks maintain reliability when considering the 'long tail' of infrequently visited websites.

Most datasets collected to test Tor WF attacks are synthetic, posing another limitation in

previous work, as these attacks may not perform well against users with varying configurations, network conditions, and browsing behaviors. Moreover, users may visit websites that are unknown to a WF attacker, including website subpages. To address this concern, we test our WF precision optimization techniques on a dataset of *genuine* Tor traffic [9] and show that many of the websites can be accurately identified in the real world.

Since flow correlation and stepping-stone intrusion detection have similar challenges in terms of identifying correlations with very low base rates, we also improve upon state-of-the-art techniques to further improve precision and reduce the impact of false positives.

Next, we introduce a WF defense that uses common patterns in web browsing traffic to regularize traffic traces, making them more difficult to distinguish. While this defense is effective, it delays traffic and is not designed to withstand flow correlation attacks. Consequently, we develop another defense to reduce the performance of both WF and FC attacks by strategically inserting dummy traffic to obfuscate the underlying traffic fingerprint.

1.1 Thesis Statement

This dissertation supports the following thesis:

Website fingerprinting and flow correlation based on deep learning pose a real threat to Tor users; however, deep learning can also help mitigate the risk by informing traffic analysis defenses.

To demonstrate this, we first improve upon existing website fingerprinting and flow correlation attacks so that they perform well, even in challenging settings. Furthermore, we show that they achieve the necessary precision to operate reliably amidst a high volume of non-targeted traffic. Then, we introduce two traffic analysis defenses. One regularizes web browsing traffic to reduce the performance of WF attackers, while the other mitigates both website fingerprinting and flow correlation without introducing delays to real traffic.

1.2 Contributions and Outline

- Prior work has shown that WF attacks can achieve high accuracy but often struggle against proposed defenses. We find that current attacks can be significantly enhanced to improve performance against some defenses. Specifically, we aim to combine data

augmentation, feature representation, and improved deep learning model architecture to bolster performance against website fingerprinting defenses.

- Tor developers have frequently criticized existing WF attacks for testing performance in unrealistic settings. For example, limited world sizes in experiments may exaggerate the performance of attacks that do not scale well in real-world scenarios. Similarly, using synthetic datasets that assume specific configurations and user behavior may lead to unrealistic attack performance. In response, we employ 'precision optimization' techniques to enhance the attacker's ability to maintain performance in large open-world settings. We further validate our techniques using a realistic dataset of genuine Tor traffic to confirm their applicability.

- End-to-end flow correlation attacks can be difficult to carry out in practice due to the low base probability (e.g. $\frac{1}{N^2}$ for N flows) that a randomly selected pair of flows is actually correlated. While the state-of-the-art DeepCoFFEA [10] flow correlation attack performs well, we demonstrate how performance can be further improved to minimize false positives, even when facing potential flow correlation defenses. Then, we test a modified version of DeepCoFFEA in the stepping-stone intrusion detection setting and find that it likewise exhibits high performance.

- While many WF defenses have been proposed, most are either ineffective against improved attacks, require additional infrastructure, or incur high latency or bandwidth overhead. Thus, there is a clear need for improved defenses to effectively protect Tor users. To address this need, we present the RegulaTor defense, which takes advantage of common patterns in web browsing traffic to regularize traffic, making it difficult for attackers to recognize traffic traces.

- While RegulaTor is effective against WF attacks, it does not defend against FC attacks, and latency-sensitive users may require a defense that does not delay traffic. Accordingly, we design a new defense, DeTorrent, that trains a long short-term memory network to generate dummy traffic insertion strategies that are resistant to adversarial retraining. DeTorrent performs well against both WF and FC attacks without delaying traffic.

Chapter 2

Background

2.1 Tor

Tor is a low-latency anonymity network that operates by encrypting traffic and routing it through a series of volunteer-run relays. In each Tor connection, or "circuit," each relay is only aware of the immediate predecessor and successor in the circuit, but not the entire path. This layered approach ensures that no single point in the network can deduce both the origin and destination of the data. Tor's architecture employs randomly selected 'entry', 'middle', and 'exit' nodes. The entry node knows the user's IP address but not their activity or final destination, while the exit node knows the destination of the data but not its origin. The encryption strategy, known as onion routing, is characterized by messages encapsulated by multiple layers of encryption, where a layer is "peeled" away at each of the intermediate "hops."

In practice, millions of users [11] use Tor for a variety of purposes, including personal privacy, censorship evasion, and political activism. However, even though Tor packages traffic into regularly sized "cells," connections are susceptible to traffic analysis attacks, such as website fingerprinting and flow correlation. As a result, there is an active field of research demonstrating these attacks as well as ways to defend against them.

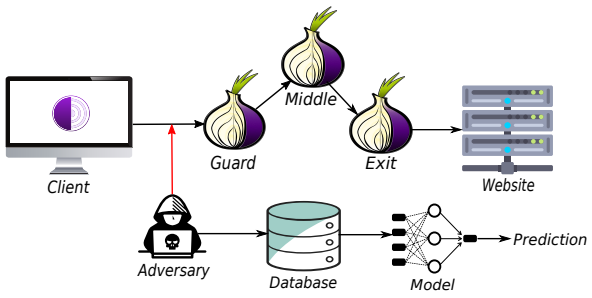

Figure 2.1: Attacker position and model creation for a WF attack on Tor

2.2 Website Fingerprinting Attacks

For WF attacks, the threat model is a passive, local adversary who eavesdrops on network traffic. Potential attackers include ISPs, network administrators, and the guard node of the Tor network, as shown in Figure 2.1. However, the attacker is not able to modify traffic in any way. While data sent through Tor is encrypted, traffic metadata such as packet timing and direction may indicate the destination of a user's traffic. To carry out the attack, the attacker first collects the traffic trace associated with the target's web browsing. Then, the attacker compares the trace to a dataset of (trace, web page) pairs, finding the matching web page based on a chosen set of features.

Hintz et al. were the first to demonstrate the risk of website fingerprinting attacks, using file transfer sizes to distinguish web pages [12]. Tor was initially thought to be WF-resistant due to its use of constant-sized cells and inherent network delays. However, researchers later began using packet volume and timing metadata to conduct the attack against Tor traffic [13, 14]. Later improvements came in the form of dataset collection and feature engineering [15], the use of more effective machine learning techniques such as k-nearest neighbors and random forests [16, 17], and the use of the cumulative representation of the traces [18].

More recent techniques leveraged deep neural networks to substantially increase WF attack performance [19, 8, 20, 5]. Most importantly, Deep Fingerprinting by Sirinam et al. [19] managed to defeat the WTF-PAD defense while pushing closed-world accuracy above 98%. Then, Rahman et al. [8] added timing information to the Deep Fingerprinting architecture to develop the Tik-Tok attack [8], further increasing performance with especially strong results against the Walkie-Talkie defense [21]. Other attacks based on deep learning include Automated Website

Fingerprinting by Rimmer et al. [22], which collected a large dataset and trained three models: stacked denoising autoencoders, convolutional neural networks, and long short-term memory networks. In addition, Oh et al. presented p-FP [20], which demonstrates that deep neural networks can be used to generate feature vectors to improve previous WF attacks in a variety of settings, and Bhat et al. released Var-CNN [23], which achieves high accuracy while using a reduced dataset.

Other efforts to improve attacks include the use of outside sources of information to reduce false positives [24] or the training of WF attacks in more challenging and data-limited settings [5, 7]. Cherubin et al. also trained a WF attack in an online setting using data safely collected from a Tor exit relay [2]. Their results demonstrated that WF attacks were feasible while using a small monitored class, but potentially infeasible with larger sets of monitored websites.

2.3 Website Fingerprinting Defenses

WF defenses alter web traffic to reduce the attacker's ability to identify the website a user is visiting. Some early defense designs, such as BuFLO and Tamaraw, operate by sending packets at a set rate and inserting dummy packets when no real data is available, making traffic associated with different websites look identical [25, 26, 27]. However, these defenses incur prohibitively high latency and bandwidth overheads.

Other defenses operate by transforming traffic such that it appears to be associated with a different website or groups of websites, thus tricking the WF attacker. These techniques include Traffic Morphing [28], Decoy pages [14], Supersequence [16], and Glove [29]. Additionally, the Walkie-Talkie defense [21] modifies the browser to operate in half-duplex mode while modifying burst sequences.

Additional latency overhead is harmful to the user experience, so some defenses have taken the approach of avoiding traffic delays. WTF-PAD [30] uses adaptive padding and distributions of typical inter-packet delays to fill the large gaps in traffic that may leak information about the trace destination. FRONT [31], on the other hand, focuses dummy padding on the beginning of the trace while randomizing dummy packet volume and timing information. Additionally, the Spring and Interspace [32] defenses operate using the Tor circuit padding framework [33], which defines state machines controlling when dummy traffic should be inserted. The state machines are designed using genetic algorithms with a fitness function based on the defense's ability to

defend against the Deep Fingerprinting attack.

Some defenses use adversarial techniques to defend traffic. Mockingbird [34] generates adversarial traces that aim to reduce the attacker's adversarial retraining accuracy by randomly perturbing in the space of viable traces. While this approach appears promising, it cannot be implemented live, as it requires knowledge of future traffic. Nasr et al. [35] propose a technique using blind adversarial perturbations to defeat DNN-based traffic analysis. Essentially, this approach trains DNN adversarial traffic generators that aim to reduce the performance of WF and FC attacks. Surakav [36] uses a Wasserstein GAN [37] to generate traffic burst volumes for a trace, where a burst is defined as a series of packets going in the same direction. Then, Surakav shapes the real traffic bursts to match that of the generated traffic.

The TrafficSliver [38] and Multihoming [39] defenses split traffic among different guard nodes such that the individual sub-traces are difficult for a WF attacker to classify. Specifically, Multihoming users connect through different access points (such as Wi-Fi and a cellular network) and merge the traffic at a multipath-compatible Tor bridge. Alternatively, TrafficSliver either splits traffic over multiple entry relays and merges traffic at the middle relay or creates multiple circuits and requests different HTTP objects over the circuits. Both TrafficSliver and Multihoming can be implemented with minimal bandwidth and appear to be effective WF defenses. However, they do not guard against all attacker types: TrafficSliver only prevents WF attacks that take place at the guard node, and Multihoming does not protect against local attackers who can see outgoing traffic.

While most defenses are evaluated and compared by testing their ability to degrade the performance of WF attacks, this methodology is limited in the sense that defenses may have weaknesses not exposed by current attacks, and that attacks may be continually improved to undermine existing defenses. To provide better estimates of defense performance, Cherubin et al. [40] present a technique for using the Bayes error rate to bound an attacker's performance given a set of crafted features. Similarly, Li et al. [41] measure the mutual information of different traffic features to estimate the amount of information leaked by various defenses. Most recently, DeepSE-WF [42] measures both Bayes error and mutual information while using learned latent feature spaces.

Other researchers have developed general frameworks supporting the design and implementation of WF defenses. For example, the QCSD framework [43] uses the deployment of QUIC

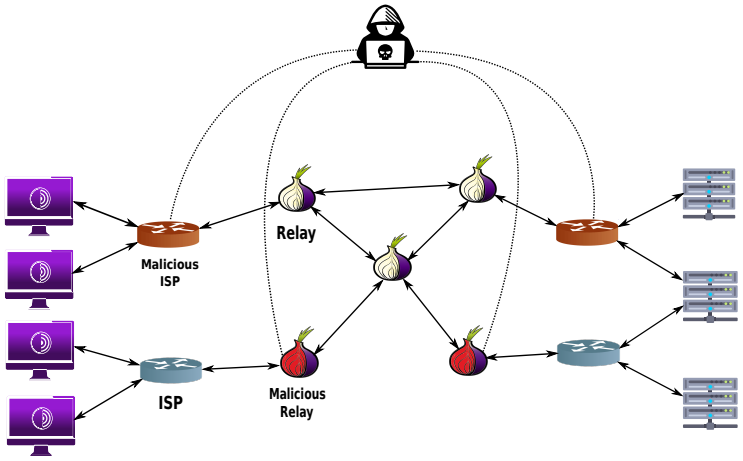

Figure 2.2: Possible attacker positions for a FC attack on Tor

and HTTP/3 to pad and shape traffic without requiring server-side modifications to support the defense. The Maybenot framework [44], on the other hand, expands the Tor circuit padding framework [33] to support a wider range of defenses, including those that delay cells.

2.4 Flow Correlation

Early discussion of security concerns in low-latency anonymity networks acknowledged the possibility of timing analysis to correlate flows, determine if two mixes are on the same path, or simply confirm that two users are online at the same time [45, 46, 47, 48, 49, 50]. While initial attack formulations included techniques such as 'packet counting' [45], later approaches turned to statistical correlation metrics such as the cross-correlation between binned packet volumes [51, 47], distance functions using mutual information and frequency analysis [52], or Spearman's rank correlation coefficient [53]. Defensive techniques proposed in response include defensive dropping [47], adaptive padding [51], and constant-rate sending [47]. However, constant-rate sending is costly in terms of bandwidth and latency overhead, while defensive dropping has been proven ineffective [51].

The original Tor design admits vulnerability to end-to-end timing correlation, though the attacker must watch traffic at both the initiator and responder to carry this out (see Figure 2.2) [48]. As a result, reducing the chance that an adversary is positioned to capture both entry and

exit flows is a concern for Tor researchers and developers. In response, several research efforts address these problems by improving Tor's guard and relay selection [54, 29, 55, 56, 57, 58].

A related threat model is the possibility of an adversary running multiple Tor relays with the expectation that a targeted client will eventually use a controlled relay as both the entry and exit. This adversary may improve their odds by preventing the creation of circuits that they're a part of, but cannot yet compromise [59]. Currently, the Tor Project's response is to have users re-use entry guards for a set time period to reduce the cumulative risk of eventually choosing a compromised circuit [60]. See Figure 2.2 for an illustration of a flow correlation attack that can make use of either relays or internet infrastructure.

Even if an adversary can capture entry and exit flows, correlating them is nontrivial due to Tor's use of fixed-sized cells and varying circuit conditions that may alter flow characteristics. Furthermore, the base rate fallacy is a concern while carrying out a flow correlation attack due to the number of potential flow combinations: to deanonymize a set of N entry and exit flows, N^2 possible combinations of flows must be evaluated. This is both computationally and practically difficult, as the number of false positive correlations will likely outnumber the number of true correlations unless the false positive rate is relatively low.

However, these challenges have been remedied in recent work by Nasr et al. and Oh et al. with their DeepCorr and DeepCoFFEA attacks [61, 10]. DeepCorr first used a deep neural network to learn how Tor circuits transform traffic, correlating flows with a lower false positive rate while using less data than previous attacks. DeepCoFFEA provides further improvements by training feature embedding networks to represent the entry and exit flows in a low-dimensional space such that correlated flows are close to one another. DeepCoFFEA also further reduces the false positive rate by dividing flows into windows and voting across the windows where the window embeddings are similar. These attacks essentially demonstrate that flow correlation is quite feasible for attackers to effectively carry out given the ability to intercept entry and exit flows.

More recent work includes a methodology for improved data collection setup in the flow correlation setting by Rimmer et al. [62]. Specifically, they propose the use of multiple geographically distributed servers while collecting data in order to improve the realism of traffic timing. Additionally, Lopes et al. [63] developed a flow correlation attack on onion services that, like DeepCoFFEA, uses sliding windows to find similarities between windows of traffic,

albeit while using a substantially different approach to find matches.

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural network particularly effective for processing data organized as a grid, such as images and video. CNNs use convolutional layers that apply a series of filters to the input to create feature maps, which highlight various features of the input data. Typically, these layers are used alongside pooling layers, which reduce the spatial size of the representation, helping to reduce overfitting. In the context of website fingerprinting, CNNs can process the timing and directionality of packet sequences in Tor traffic to effectively detect distinct patterns associated with different websites.

2.6 Long Short-Term Memory networks

Recurrent neural networks (RNNs) are a class of artificial neural networks designed for processing sequential data. Unlike traditional feedforward networks, RNNs have internal loops that allow information to persist, enabling them to use previous steps' outputs as inputs for the current step. While RNNs can theoretically consider longer-term input dependencies in sequential data, they suffer from the 'vanishing gradient' problem, where back-propagated gradients tend to either zero or infinity. To address this, Long Short-Term Memory (LSTM) networks add a set of 'gates,' called the input, output, and forget gates, that regulate information across cell states. Consequently, the LSTM is better able to consider long-term dependencies, giving it a 'long short-term memory.' LSTMs also support the use of irregularly-spaced time series of variable length, as they can make repeated decisions throughout the time series.

2.7 Transformers

The Transformer, as introduced by Vaswani et al. [64], is a deep learning model known for its ability to understand contextual relationships in sequential data. Prior to transformers, RNNs and their variants, LSTM networks and gated recurrent units (GRU) networks, were the dominant architectures for sequence-based tasks. While effective, these models process sequences

step-by-step, making it difficult to handle long-range dependencies and resulting in increased computational times for long sequences.

At its core, the Transformer employs a self-attention mechanism. This mechanism calculates a context-enriched representation for each input token by computing a weighted sum of all tokens. The weights are derived from the relevance of each token to others, determined through a process involving *query*, *key*, and *value* representations. Specifically, relevance is assessed by the dot product of *query* and *key* followed by a softmax function, influencing how the *value* representations are weighted.

2.8 Preliminaries

2.8.1 WF Attack Settings

WF attacks are typically tested in two different settings: *closed-world* and *open-world*. In the closed-world setting, we assume that the target visits one of a defined set of web pages, and the attacker simply has to determine which of those web pages was visited. Because a real-world target may visit any number of web pages, this setting is unrealistic. Still, the closed-world accuracy is useful for comparing and evaluating defenses against a relatively strong WF attacker.

On the other hand, the open-world setting provides a much more realistic setting where the target can visit any number of web pages. In this case, the attacker's goal is to determine whether or not the target visited a *monitored* web page using information from the packet trace. While this task may be significantly more difficult given that the attacker cannot possibly train a model on all possible web pages, it is much more similar to the real-world scenario of an adversary trying to catch users visiting censored web pages.

In the closed-world setting, the evaluation is straightforward, as the attacker simply needs to determine the correct website with the highest possible accuracy. However, the open-world setting is more complicated due to the imbalanced classes, as the attacker may only be able to monitor a small portion of web pages a user could visit. Accordingly, precision is a much more useful metric for evaluating the usefulness of a WF attack in a realistic scenario. Alternatively, recall, representing the fraction of monitored web pages that were retrieved, is also used in the open-world setting to demonstrate the sensitivity of an attack. An ideal attack has both high

precision and high recall; however, one often comes at the expense of the other.

2.8.2 Defense Overhead

WF defenses have the potential to impact user experience and increase strain on the Tor network by increasing latency, delaying page loading, and increasing the required bandwidth. As a result, we evaluate the *latency overhead* and *bandwidth overhead* of each defense. We find latency overhead by calculating the additional time required to send the *defended* trace compared to the undefended trace and dividing by the time required to send the original trace. In practice, we do this by subtracting the sending time of the last real packet in the defended trace from the sending time of the last packet in the undefended trace. Intuitively, the latency overhead metric aims to indicate how much longer the user will have to wait to load a web page.

The bandwidth overhead is found by dividing the number of dummy packets sent in the defended trace by the number of packets in the undefended trace. Essentially, it represents how much more data will be transmitted while loading a web page with the WF defense enabled.

Though increased bandwidth may strain the Tor network, discussion in the Tor community indicates that it is preferable to latency overhead. This is because a reasonable degree of bandwidth overhead can be managed by Tor without affecting the user experience, but increased latency would increase page load time and potentially contradict Tor's position as a 'low latency' protocol [33].

Chapter 3

Convolutional vision Transformer WF Attack

A variety of Tor website fingerprinting defenses threaten to diminish the impact of leading attacks [31, 65, 38, 39, 43]. Nonetheless, while adhering to the adage that "attacks only get better," we investigate how a website fingerprinting attacker may still compromise user privacy despite these defenses. Accordingly, we focus on testing an improved WF attack, CvT, against traffic that has been obfuscated by effective WF defenses.

These improvements can take the form of model architecture, data augmentation, changes in how the raw data is represented as input for the model, and changes to reflect best practices in deep neural network training, such as learning rate warm-up and decay. In general, past WF attacks have applied advancements in data analysis and machine learning to improve the state of the art; we continue this trend by applying attention mechanisms and other best practices.[1]

3.1 Related Work

While the full background of website fingerprinting attacks is best described in Chapter 2, some attacks in recent years have likewise aimed to develop WF attacks that perform well in realistic

[1]Content from this chapter has been adapted from "LASERBEAK: Evolving Website Fingerprinting Attacks with Attention and Multi-Channel Feature Representation" which is currently under review for publication in IEEE Transactions on Information Forensics and Security [66]

and challenging settings. For instance, to address the resource-intensive web crawling needed to train deep learning-based WF attacks, the Triplet Fingerprinting attack [5] uses triplet networks for N-shot learning, which is when a machine learning model is trained to make predictions based on a small number of examples. To test how well the attack is able to generalize, the authors then evaluate the approach with training and testing sets collected years apart, among other challenging settings. Additionally, the GANDaLF approach [7] takes a limited set of training data and uses generative adversarial networks (GANs) to generate a larger, synthetic dataset and thus improve WF performance.

Cherubin et al. [2] then adapt the Triplet Fingerprinting attack to an online setting where the model is trained and tested on (safely collected) genuine Tor traffic. Given the challenging setting, they find that the approach works well when the monitored set is small, but experiences reduced performance for larger monitored sets.

Like CvT, the Robust Fingerprinting [6] attack focuses on investigating new feature representations and undermining WF defenses. In particular, they use a Traffic Aggregation Matrix (TAM), which separately bins the upload and download traffic volumes over time, to represent the traces. This representation captures important information about each trace while remaining resistant to traffic obfuscation, such as dummy traffic and artificial delays.

3.2 Model Architecture

The Convolutional vision Transformer (CvT), as proposed by Wu et al. in 2021 and illustrated in Figure 3.1, seeks to enhance the performance and efficiency of vision transformers by integrating convolutional layers. This combination is aimed at harnessing both the robustness of convolutions in handling shifts and scale changes and the transformer's ability to process long-range dependencies. This dual approach allows the CvT to effectively analyze both local and global structural aspects. Additionally, the convolutional layers aid in identifying and understanding hierarchical patterns.

Structurally, the CvT adopts a multi-stage hierarchical arrangement similar to traditional vision models. At each stage's beginning, a convolutional layer is employed to expand the feature dimensions while simultaneously reducing spatial resolution through strided convolutions. As one moves through these stages, the feature dimensions grow, but the sequence length decreases, making the attention mechanism more computationally efficient.

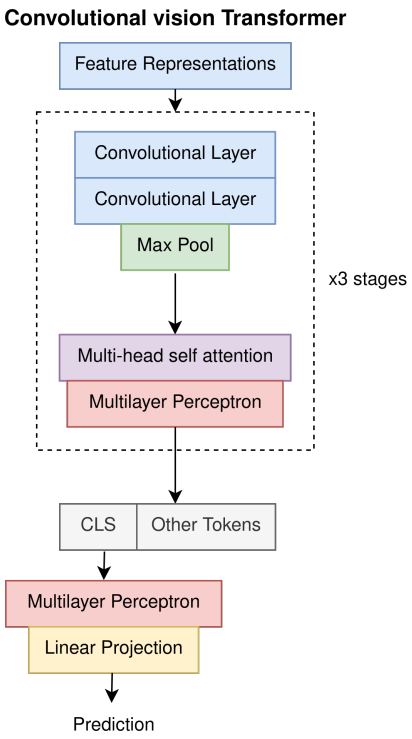

Figure 3.1: Adapted Convolutional vision Transformer for WF

Moreover, CvT innovates by replacing the typical linear projection method, used to generate the 'key,' 'query,' and 'value' matrices in transformers, with a convolutional projection technique. This change serves a dual purpose, using convolutional kernels to capture and integrate local interactions within the attention mechanism while applying strided convolutions to shorten the 'key' and 'value' sequences. Attention is computed in this condensed space, further enhancing the model's computational efficiency.

To adapt the original Convolutional vision Transformer for website fingerprinting, we replace the original linear convolutional layer with a structure inspired by the convolutional block in Deep Fingerprinting (DF), as per Sirinam et al. [19]. This new layer consists of one-dimensional convolutions, batch normalization, ReLU activations, max pooling, and dropout for regularization. The full CvT model thus processes the input by alternating between the DF-based layers and the transformer-type layers. Eventually, a classification token, which is created by taking

into account the embeddings of the other tokens, is used along with a final linear layer to predict the class of the input trace. Overall, these modifications to the CvT architecture result in marked performance improvements, particularly against defended datasets.

3.3 Feature Representation

Another significant improvement is the use of multiple different representations of the same trace. Previous WF attacks have frequently only used one channel, such as the direction-only features, to represent a trace. However, each input channel of the first convolutional layer in the CvT model consists of a different representation of the input trace.[2] These representations include the cumulative number of packets at different points in time, the inter-arrival times (which can be multiplied by the +1 and -1s representing packet directions), the logarithm of the inverse of the inter-arrival times, and the weighted average of the number of packets being sent, among others. Summaries of these feature representations, among others used in testing, are shown in Table 3.1. Since most website fingerprinting defenses also fail to obfuscate the quick 'bursts' of traffic, where packets are separated by less than one millisecond, we also find it useful to have one feature representation that contains only this type of traffic. Typically, few of the remaining traffic packets in this representation will be dummy packets, making fingerprinting defended traffic more feasible.

As a result, the CvT attack can use the varied feature representations to focus on different aspects of the input trace. This aligns with past work that uses mutual information estimates to demonstrate that different hand-crafted feature representations leak varying amounts of information in the website fingerprinting setting [41]. We find that using three to four different feature representations maximizes results; including further representations tends to decrease performance by increasing the model's propensity to overfit.

3.4 Data Augmentation

One of the WF attacker's advantages is that they can retrain their attacks in response to a newly implemented traffic analysis defense. Multiple previous works have discussed this when

[2]The idea of using several feature represents to improve WF attack performance, as well as several of the feature representations shown in Table 3.1, was developed in tandem with Nate Matthews.

Name	Description
directions	Packet directions represented as +1 for upload and -1 for download
times	Raw packet timestamps
directional time	Packet timestamps with directional information (times × directions)
cumulative	Total number of packets sent so far calculated at regularly-spaced intervals in the trace
inter-arrival times	Gaps (in seconds) between the packets in the trace
bursts only	Times of packets that are within one millisecond of another packet in the original trace
inverse inter-arrival times	Inverse of gaps between packets
log inverse inter-arrival times	Modified inter-arrival times to emphasize the differences between small values. Uses logarithm base 10 of the inverse of the inter-arrival times. Created in response to the nearly power-law distribution of gaps between packets
directional log inverse inter-arrival times	Logarithm base 10 of the inverse of the inter-arrival times with encoded packet direction
running rates	Weighted average of the number of packets being sent in the trace at different points in the download. Weight can be tuned to consider recent traffic more or less heavily

Table 3.1: Feature Representations used as input for deep learning traffic analysis models

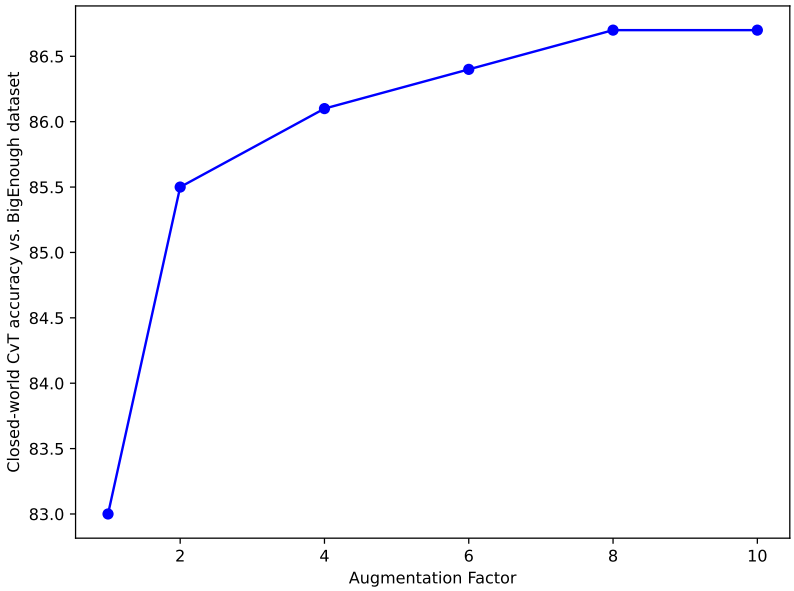

Figure 3.2: CvT closed-world performance against data-augmented BigEnough dataset. Note that performance is not as high demonstrated in the final CvT vs BigEnough experiment, which included additional CvT tuning.

evaluating defenses [67, 68, 69]. While some defenses attempt to use randomness to confuse attackers, either with dummy padding placement or in terms of how traffic is split, attackers can mitigate their performance losses with data augmentation techniques.

Specifically, a WF attacker can simulate a WF defense multiple times on each collected trace to help the attack model better generalize and 'understand' the varied defense strategies each defense may use.[3] When evaluating CvT and the other attacks in the closed-world setting, the defenses were simulated 10 times each on each trace. We find that increasing the data augmentation multiplier past 10 does not yield significantly higher performance, as shown by Figure 3.2. For each defense simulation, any randomly set parameters were re-selected, presenting a new way in which the base trace might be defended.

[3]The approach of using data augmentation techniques to improve attack performance against WF defenses was developed in a collaboration with Nate Matthews.

3.5 Training Details

3.5.1 Training Improvements

In addition to the data augmentation and feature representation modifications, we also make improvements to our model training. In particular, we use the AdamW optimizer [70], which better incorporates weight decay. We also use learning rate warmup and cosine decay [71, 72] to improve training stability, as these appear to be current best practices. Furthermore, we use label smoothing, which is where we use values slightly less than 1 for the true class and slightly more than 0 for other classes. This prevents models from becoming too confident in their predictions, reducing the risk of overfitting [73].

3.5.2 Datasets

To benchmark CvT, we use the BigEnough (BE) dataset, which was collected in late 2021 for an evaluation of multiple website fingerprinting defenses by Matthews et al. [68]. This dataset contains data from 95 websites, where each website is represented by 20 samples from each of the 10 website subpages. Including multiple subpages for each website increases the intra-class variance, making this dataset more challenging. While the original dataset collected traces in the Standard, Safer, and Safest security configurations for the Tor Browser Bundle, we only use traffic from the Standard setting. The websites were chosen from the most popular websites in the Open PageRank initiative and the collection methodology is based on that of the GoodEnough dataset collected by Pulls et al. [65].

We also test CvT against the open-world dataset collected by Gong et al. using WFdefproxy [74] to test the Surakav defense. We refer to this as the Gong-Surakav dataset. It consists of 10k samples from 100 websites as well as 60k unmonitored samples for each of the defense settings, which includes undefended, FRONT, Surakav, and Tamaraw.

3.5.3 Model and Experiment Details

The adapted CvT model is somewhat larger than many previously published WF attacks, with 6.65 million parameters compared to about 3.95 million parameters for the Tik-Tok attack [8], based on the author's provided code. We train it with a maximum learning rate of 10^{-4} for 250 epochs on an Nvidia GeForce RTX 3090 GPU using a framework written in PyTorch. The

recorded memory usage is about 1 GB for the undefended BE dataset experiment with a batch size of 32, though memory usage can be much larger when testing CvT against data-augmented defended datasets.

For the closed-world experiment using the BE dataset, we use 10 multisamples for each trace. However, because the Gong-Surakav dataset defenses are generated with WFdefproxy [74] rather than simulated, we do not perform data augmentation in the open-world setting. Because the average trace size in the BE dataset is over 4,000 and the defenses often add a high volume of dummy traffic, we use a maximum trace input size of 10,000. The specific set of feature representations used in the evaluation below includes: **directional time**, **normalized cumulative**, **normalized times**, **directions** $\times$ **inter-arrival times**, **log inverse inter-arrival times**, and **running rates**. Note that the normalized times and normalized cumulative representations are just normalized versions of the raw times and cumulative representations.

In our evaluation, we use the data augmentation approach for all tested WF closed-world attacks. As a result, the true gap between our techniques and the prior WF attacks as they were published may be larger than our results indicate. We split both the BE and Gong-Surakav datasets so that 90% of the examples are used for training and the rest are used for testing. In the open-world setting, we present our results in terms of precision and recall, setting the confidence threshold to illustrate the trade-off between them. This confidence threshold operates so that a determination that a trace is associated with a monitored set class is only kept if the model output is greater than the set threshold.

3.5.4 Defense Configurations

To fairly test the defenses, we configure them to best defend the traffic in the BigEnough dataset. For FRONT, we use the FT-2 configuration, which is more effective than FT-1 but incurs a higher bandwidth overhead. Similarly, for RegulaTor, we adapt the RegulaTor-Heavy defense to account for the increased traffic volume in the BigEnough dataset. The Interspace defense is simulated using the provided Tor padding machine. Because the BigEnough dataset traces are similar to the GoodEnough dataset [65] used to test Interspace, we do not modify the padding simulator parameters. To simulate the traffic splitting datasets, we choose the most effective strategy, batched weighted random, and evaluate it using either two or five substreams.

Similarly, the defended Gong-Surakav dataset adjusts the FRONT and Tamaraw defenses to

	Defense	Total Overhead	
		Bandwidth	Time
BigEnough	FRONT	50%	0%
	Interspace	96%	0%
	RegulaTor	92%	7%
	TS-2	0%	0%
	TS-5	0%	0%
Gong-Surakav	FRONT	80%	0%
	Tamaraw	85%	38%
	Surakav-light	44%	24%
	Surakav-heavy	63%	23%

Table 3.2: Bandwidth and time overhead for each defense

account for the higher observed bandwidth. For Surakav, both the light and heavy versions of the defense are evaluated. For a summary of the defense overheads, see Table 3.2.

		BigEnough Dataset [75]				
Model	Undef.	FRONT	Inter.	R.Tor	TS-2	TS-5
DF	95.4	76.0	65.1	14.7	4.4	2.3
Tik-Tok	94.8	81.8	76.5	26.5	65.6	38.8
CvT	95.6	94.4	92.7	39.8	71.9	37.2

Table 3.3: Closed-World performance on BigEnough dataset

3.6 Demonstration

3.6.1 Closed-World Performance

In order to best demonstrate the capabilities of an attacker, we generate the defended datasets by simulating the defense 10 times for each original training sample. This technique, seen in prior work, improves the attack model's ability to generalize and learn how the defense is implemented [65, 75]. We also train Deep Fingerprinting (DF) [19] and Tik-Tok using the parameters defined in the original implementation.

While CvT performs only slightly better than existing attacks against the undefended dataset, as shown in Table 3.3, it achieves much higher accuracy against the FRONT, Interspace, and RegulaTor defenses. In fact, its accuracy against the FRONT and Interspace defenses is nearly as high as its accuracy against undefended traffic. Against RegulaTor, CvT still performs well given that the defended traces have been highly regularized, making many of them particularly difficult to identify. CvT's performance against TrafficSliver, however, does not necessarily exceed that of Tik-Tok.

In general, the Tik-Tok attack is reasonably effective against all of the defenses, with higher accuracy as a result from using multiple simulations for each original trace. Deep Fingerprinting is similarly effective against most defenses, except for TrafficSliver. We suspect that DF's lack of timing information causes it to struggle to identify 'slivers' of traffic that have been split from larger traffic flows.

3.6.2 Open-World Performance

In the open-world, CvT is compared to the state-of-the-art WF attack Tik-Tok [8] on the relatively challenging Gong-Surakav dataset. The results are shown in Figure 3.3.[4] We find that CvT's and Tik-Tok's performance is very similar against undefended traffic, which is not particularly surprising given that CvT only marginally outperformed DF against undefended traffic in the closed-world setting. However, CvT demonstrates its effectiveness against FRONT, Surakav-light, and Surakav-heavy defenses, where it manages to outperform Tik-Tok. This is particularly obvious against FRONT and Surakav, where Tik-Tok struggles to learn to reliably

[4]This figure was adapted from a paper by Matthews et al. [66], which contains the results of a collaboration aimed at producing more effective WF attacks.

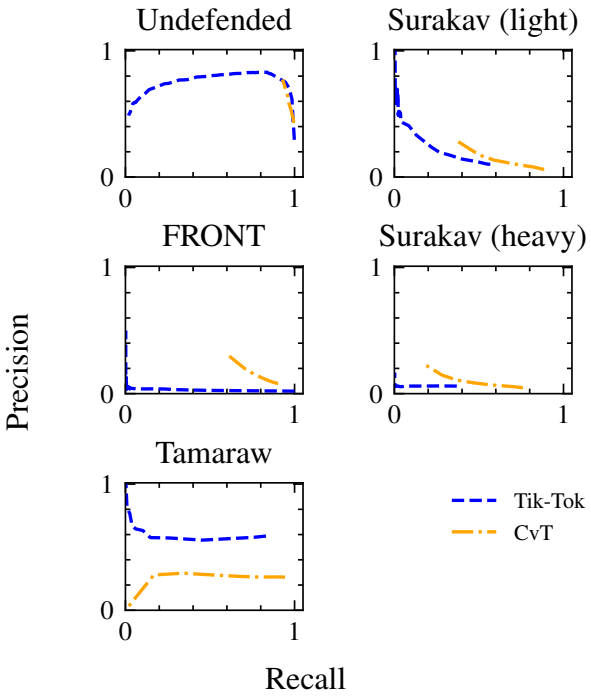

Figure 3.3: CvT vs Tik-Tok performance on the Gong-Surakav dataset, with a base rate of 2%

identify any websites. However, Tik-Tok then outperforms CvT against the Tamaraw defense, identifying nearly all of the monitored set with reasonable precision.

Overall, it appears that the larger CvT model, along with the feature representation and training improvements, help to improve its performance against defended datasets. On the other hand, the size of the CvT model as well as the often redundant feature representations make CvT prone to overfitting, which can harm performance in some settings.

3.7 Summary

This chapter explores the efficacy of a modified Convolutional vision Transformer architecture for Tor website fingerprinting attacks, particularly against defended datasets. CvT integrates convolutional layers with an attention mechanism to leverage both local and global structural information, enhancing performance and computational efficiency.

To adapt CvT for website fingerprinting, we make modifications inspired by Deep Fingerprinting (DF), including one-dimensional convolutions, batch normalization, and ReLU activations. The updated model also utilizes varied feature representations of input traces, such as packet directions, timestamps, cumulative packets, inter-arrival times, and log inverse inter-arrival times, to improve attack accuracy. Furthermore, data augmentation through multiple defense simulations per trace further increases model generalization.

CvT demonstrates superior performance against several defenses, including FRONT and Interspace, compared to existing models like DF and Tik-Tok. Despite occasional overfitting, CvT's comprehensive architecture and robust training practices yield notable improvements in identifying defended traffic.

Chapter 4

Website Fingerprinting Precision Optimization

In 2020, Wang et al. introduced a method for enhancing precision in open-world scenarios termed "Precision Optimizers" (POs) [76]. These POs work in addition to a primary model to filter out uncertain predictions. However, this approach did not extend POs to deep-learning models. Our approach, inspired by Wang et al.'s methodology, is tailored for deep learning models. The rationale behind prioritizing precision aligns with Wang et al.'s argument: high precision is more critical than high recall. This is due to factors like the likelihood of users revisiting websites, the need for attackers to address the base rate fallacy, and the tendency for privacy-aware users to restrain their online activities even under the threat of low recall attacks [76].[1]. In particular, we find that the *confidence*, *k-neighbors*, and *ensemble* approaches can be successfully adapted to improve the precision of WF classifications made by deep learning models.

[1]Content from this chapter has been adapted from "LASERBEAK: Evolving Website Fingerprinting Attacks with Attention and Multi-Channel Feature Representation" which is currently currently under review for publication in IEEE Transactions on Information Forensics and Security [66]

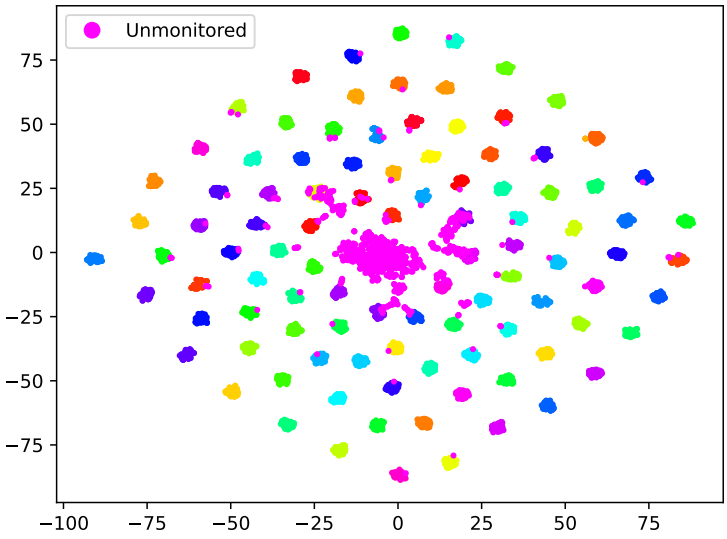

Figure 4.1: Undefended Gong-Surakav test set embeddings, visualized using t-SNE. Monitored traffic is shown with a variety of colors based on the class of each trace.

4.1 Related Work

In past literature, WF attacks have often been considered difficult to scale well enough to operate in the real world. For example, a 'Critical Evaluation' of WF attacks [3] published in 2014 argues that prior work makes potentially unrealistic assumptions about user settings while also neglecting to consider the *base rate* of visited to monitored pages. More precisely, it is possible for an attack to report a high TPR and low FPR while still being unreliable, because even the low FPR may lead to a high number of false positives given a very large open world size. Given the large size of the web, WF attacks should be able to demonstrate that they can identify rarely-visited web pages with high precision.

Later work by Panchenko et al. [18] aims to carry out WF attacks at 'Internet scale' by collecting a large and realistic dataset while avoiding many of the assumptions made in prior work. Despite using state-of-the-art WF attacks, they find that none of their approaches scale well enough to identify particular web pages in large open-world settings.

Other recent work by Cherubin et al. uses genuine Tor traffic along with the Triplet Fingerprinting attack modified to operate in an online learning setting. They test their approach in a variety of different settings, finding that it functions fairly well when monitoring a small set of relatively popular websites, but experiences low performance otherwise. While their approach uses a more sophisticated attack compared to past work, we believe that we can still improve upon these results in this dissertation. For instance, we can train over a large pre-collected dataset rather than operating in an online setting and using a mean embedding vector. We also have an updated WF attack that makes use of improved architecture and feature representations.

Work by Wang et al. to improve WF attack precision in the open-world likewise found that existing WF attacks tend to struggle to reliably identify low base rate pages. However, they improve this situation by developing 'precision optimizers' that can be applied to existing WF attacks. Their approaches include the *k-neighbors* strategy, a confidence-based PO, a distance-based PO, and the *ensemble* strategy. They find that the best optimizers can greatly improve precision against low base rate pages, making WF attacks appear threatening even in the large open-world setting. These approaches also provide the inspiration for the deep-learning precision optimizers presented in this chapter.

An attack related to the *k-neighbors* approach described in this chapter is k-fingerprinting [17], which uses a set of features extracted from each trace with the k-nearest neighbors algorithm

for classification. While this attack performed well, it is likely limited by its use of manually crafted features. Another similar approach is the k-Nearest Neighbors classifier presented by Wang et al.[16], which likewise uses a set of features to create an embedding for each trace and calculate the distance from 'nearby' labeled training examples.

Another example of an ensemble-based WF attack is snWF [77], which ensembles different checkpoints of the model saved during the training process. Specifically, it encourages the model to converge to different local minima by periodically increasing the learning rate, while saving the model each time it converges. The authors find that this approach makes the attack more robust, especially when classifying websites not previously seen by the attacker.

4.2 Techniques

Confidence In the *confidence* technique, we use the fact that the model outputs probabilities for all the websites in the monitored set even during open-world classification. This technique says that a prediction of a trace being in the monitored set holds only if the model's highest predicted probability for a website is more than a set threshold greater than the second most likely website. By varying this threshold, we can tune our desired trade-off between precision and recall.

K-neighbors Inspired by the technique of the same name used by Wang et. al, this approach uses the output of a fully-connected layer near the final output of a model as an embedding of the trace. Since traces from the same class typically have similar embeddings, and the distance between traces can be used to gauge their similarity, we can determine whether a given prediction of a trace being in the monitored set is a confident one by determining whether its embedding is near that of traces in the same class. In particular, if k neighbors are in the validation set and are of the same monitored set class, then we can be confident that the trace is in the monitored set. Additionally, we can tune the choice of k to modify the trade-off between precision and recall.

An illustration of these embeddings, created using t-distributed stochastic neighbor embedding with two components, is presented in Figure 4.1. Note that the monitored set classes are closely clustered and well-separated, while the unmonitored set traces are sometimes misclassified as a monitored set class. However, the *k-neighbors* approach is generally able to determine

whether a given trace is in a 'pure' cluster, and thus make predictions with high precision.

Ensemble To create an ensemble of models, we train three models so that each takes as input a different set of feature representations. Then, we determine that a given trace is in the monitored set only if all of the models agree that the trace belongs to a given monitored class. The underlying rationale of this approach is rooted in the concept of ensemble learning, where the models offer varied perspectives on the data, effectively reducing individual biases and errors. In other words, this approach is likely to filter out borderline predictions, thus enhancing precision.

4.3 Implementation Details

4.3.1 Datasets

In this chapter, we test our techniques against the Gong-Surakav dataset[36] and the GTT23 dataset[9] collected by Jansen et al. The Gong-Surakav datasets were collected by crawling websites and implementing defenses with the WFdefproxy[74] framework. The monitored set in each Gong-Surakav dataset consists of 10k samples from 100 websites, while the unmonitored set consists of 60k samples collected from the Tranco top 1 million sites[78]. The crawl was run over a two months. Because the defenses were built into the crawler rather than simulated, we expect that these datasets capture the effectiveness of the defenses in a realistic scenario.

The GTT23 dataset, detailed in the paper "A Measurement of Genuine Tor Traces for Realistic Website Fingerprinting," addresses the limitations of synthetic datasets by capturing genuine Tor user behavior. It comprises 13,900,621 circuits to 1,142,115 unique destination domains over 13 weeks, making it the largest of its kind. The dataset includes diverse traffic types and connections to 68 unique server ports, representing real-world Tor usage more accurately. Collected using Tor exit relays, the process was guided by strict ethical and privacy protocols. GTT23 offers a realistic basis for evaluating website fingerprinting attacks and defenses, providing valuable insights into actual Tor network traffic.

4.3.2 Technique Parameters

Confidence For the *confidence* precision optimizer, we train and test the model in a manner similar to that of the base model in the open-world setting. However, during evaluation, we only 'keep' a prediction that the trace belongs to a monitored class if the output from the softmax layer for that class exceeds a set threshold above that of the next most likely class. No other modifications to the base model are required.

K-neighbors To implement the *k-neighbors* approach, we modify the underlying model architecture to better support meaningful embeddings. Against the Gong-Surakav dataset, we constrain the dimensionality of the second-to-last fully connected layer to 128. While this modification may constrain performance slightly, it also forces the model to use the limited representation size in a 'useful' manner, ensuring that the embeddings are useful for later analysis. For the GTT23 dataset, we find that performance is highest when using dimensionality 512 for the second-to-last fully connected layer. However, in both cases, performance was relatively stable given choices of dimensionality from 64 to 1024. Outside of these bounds, performance begins to decrease moderately, with a decreasing number of websites identified with high precision for recall above 0.2. In both cases, we use a model trained in the closed-world setting, meaning that it is only trained to distinguish between the identities of the monitored set traces.

Next, we use a held-out validation set, which is separate from the test set, to generate a set of embeddings labeled by their true classes. We include both the monitored and unmonitored traces to better understand which regions of embeddings can be associated with a given monitored class with high certainty and which ones are frequently confused with traces from the unmonitored set.

To test the performance of this approach, we use the modified model to produce embeddings for each trace in the test set. Then, we predict the class of each trace based on the identities of the nearest k neighbors, where the neighbors are sampled from the *validation* set. If the k nearest neighbors are not all of the same class, or if any are of the unmonitored class, then this approach predicts 'unmonitored,' as this implies that the trace cannot be classified accurately. Otherwise, the class of the nearest neighbors is predicted.

Ensemble The *ensemble* approach is very similar to that of the typical open-world evaluation, except that we train multiple open-world models using different input representations.

Against the Gong-Surakav dataset, we train three models, finding that using additional models is excessively computationally expensive for decreasing precision improvements. For example, the number of websites with precision near 1.0 for recall above 0.2 decreased from 72 for two models to 68 for three models. The first uses six different representations: **directional time**, **normalized cumulative**, **normalized times**, **inverse inter-arrival times** $\times$ **directions**, and **running rates**. Note that the normalized cumulative and normalized times are simply normalized versions of the cumulative and times features shown in Table 3.1. The second model uses the **directions**, **times**, **directional time**, **burst edges**, **cumulative**, and **normalized cumulative** representations. Here, **burst edges** is a representation introduced by a collaborator that stores the difference between adjacent values in the direction representation. The third model then uses the **normalized times**, **inter-arrival times**, **inter-arrival times** $\times$ **directions**, **log inverse inter-arrival times**, **log inverse inter-arrival times** $\times$ **directions**, and **running rates**.

Against the GTT23 dataset, we achieved the best performance using just two models in the ensemble. Beyond two models, performance declined somewhat with slightly improved precision for moderately decreased recall. The first used the **inverse inter-arrival times**, **directional time**, and **normalized cumulative** representations, while the other used the **directional**, **inter-arrival times**, and **times** representations. Using more than three feature representations appeared to decrease performance by exacerbating model overfitting.

To improve the precision of predictions, we only keep a monitored set prediction if the models unanimously predict a monitored class with softmax output above a set threshold. This threshold is then varied to test different precision-recall trade-offs.

4.3.3 Experiment Details

Against both the Gong-Surakav and GTT23 datasets, we use the base model **Laserbeak** developed by Matthews et al. While the CvT model achieved high performance, the **Laserbeak** model was developed to improve upon CvT's weaknesses, including its propensity to overfit on a small dataset. As a result, we feel that the **Laserbeak** attack is a more reliable base model for demonstrating the precision optimization techniques. The framework used to train the base model is written in PyTorch. We train the model with a maximum input representation size of 10,000 with a maximum learning rate of 2×10^{-3} and a batch size of 128. The original

datasets are split into training, testing, and validation sets. We train the models on an Nvidia RTX 3090. Because the Gong-Surakav dataset uses live implementations of the defenses rather than simulated versions, we do not perform data augmentation in this chapter. We split the Gong-Surakav dataset so that 90% of the data is used for training and 10% is used for testing.

Because we evaluate the PO precision and recall on the GTT23 dataset on a per-class basis, we then use the micro-average and macro-average precision and recall to present the overall PO performance for all of the classes. Micro-average precision and recall aggregate the contributions of all classes to compute overall metrics by summing true positives, false positives, and false negatives across all classes. Macro-average precision and recall, on the other hand, compute these metrics independently for each class and then average them, treating each class equally. This dual approach provides a comprehensive evaluation, balancing overall performance (micro-average) with per-class performance (macro-average).

We filter out the low-volume circuits in the GTT23 dataset, as many of the circuits carry very few Tor cells, making it unlikely that they are used for web browsing. We also filter out traffic not sent to ports 443 or 80 and only use traffic from the first 47 days of the data collection. During preprocessing, we considered the top 100 sites as the 'monitored' set and the rest of the websites as the 'unmonitored' set. However, during testing, we simply considered if our approach could identify a given website among the entire world of possible traffic. Overall, the monitored set has 824,345 traces and the unmonitored set has 1,278,887 traces. We then split this dataset so that 50% is used for training, 25% is used for validation and for the creation of embeddings in the *k-neighbors* approach, and 25% is used for testing. Given the power-law distribution of domains in the GTT23 dataset, some domains made up a large portion of the circuits in the monitored class, while others were associated with fewer than 2,000 circuits.

4.4 Demonstration

4.4.1 Against the Gong-Surakav Dataset

We evaluate our precision optimizers against the Gong-Surakav dataset to test Surakav and other website fingerprinting defenses [36], using a trained WF attack as the base classifier. This particular attack uses both convolutional layers and multi-head attention. Our results are

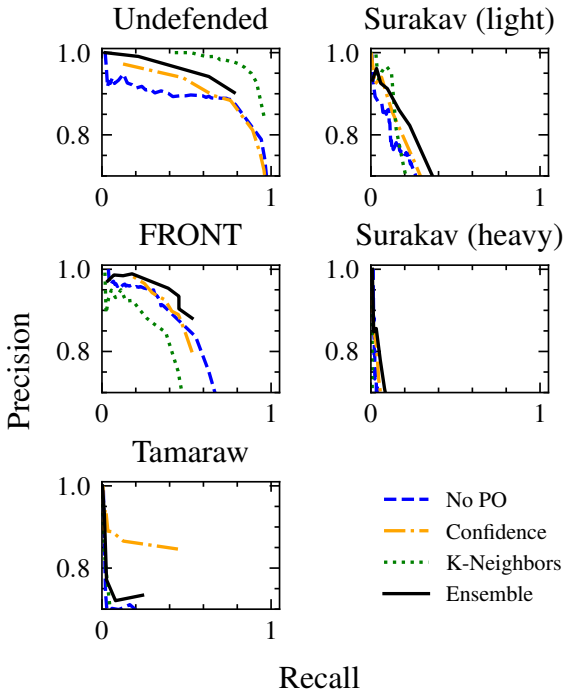

Figure 4.2: Precision optimization results compared to base WF attack

presented in Figure 4.2.[2]

We find that the *k-neighbors* and *ensemble* approaches do particularly well on the undefended dataset, with the *k-neighbors* approach achieving 100% precision with recall up to 53% and the *ensemble* optimizer achieving 99% precision at 22% recall. In both cases, the POs make the attack much more feasible in real-world settings, where the world size is dramatically larger than 50k unmonitored sites. The *confidence* optimizer is effective primarily at lower recall values, attaining 97% precision at 11% recall, for example.

The precision optimizers have varying performances on different defenses. Against FRONT and Surakav, the *ensemble* approach works well, particularly for higher recall rates. For example, against Surakav-heavy, it gets 78% precision for 6% recall, compared to the base precision of 63% for similar recall. The *confidence* optimizer has modest gains at best against FRONT and Surakav. Interestingly, it does very well against Tamaraw, improving precision to 85% for 45% recall, which is significantly better than the base attack performance of 70% precision for just 10% recall. The *k-neighbors* optimizer generally struggles to improve performance, likely because the previously well-clustered embeddings are obfuscated by these defenses.

However, the *confidence* method continues to provide reasonable performance improvements, most notably against Tamaraw, where it improves precision to 85% for 45% recall, compared to the base attack performance of 66% precision for 85% recall. Similarly, the *ensemble* approach moderately but consistently increases precision against the defended datasets. Its best performance is against Surakav-heavy, where the precision is 78% for 6% recall, compared to the base precision of 63% for similar recall.

4.4.2 Against A Real-World Dataset

Base Performance

To provide a baseline and demonstrate how effective a WF attack can be against genuine user behavior, we first test the Laserbeak attack against the GTT23 dataset. Because of the power-law distribution of circuits per domain in GTT23, using a monitored set of infrequently used websites severely limits the data available for training. However, including the most commonly

[2]This figure was adapted from a currently submitted paper containing the results of a collaboration aimed at designing more effective WF attacks.

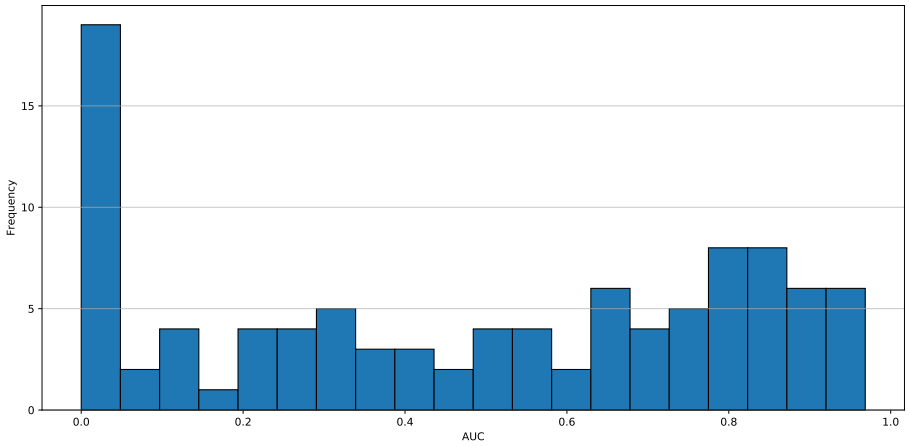

Figure 4.3: Distribution of per-class area under the precision-recall curve for the base attack

visited websites in the monitored set presents a very easy attack setting, as the resulting unmonitored set may then not be much larger than the monitored set. Using a monitored set also means that misclassified traces may still be counted as 'true positives' if they are classified as another monitored trace.

Consequently, to evaluate the attack in an appropriately challenging setting, we test the per-class precision and recall for the top 100 websites. This simulates the difficult but plausible scenario where an attacker would like to watch a user's traffic and identify the specific websites they visit. By focusing on the top 100 websites, we also ensure that GTT23 contains sufficient training data for each website. The unmonitored set then contains all of the other traffic that is collected during the set timespan; because we randomly sample traffic from GTT23, we are also confident that experiments are conducted using a realistic *base rate* of targeted websites.

First, we run the base attack and evaluate the precision and recall on a per-class basis. It appears that a substantial minority of websites are very difficult to identify, as the attack has both low precision and low recall against them (see Figure 4.3). The remainder of the websites have varying areas under the curve (AUC), with another significant minority being identified with relatively high precision most of the time. Note that an AUC of 1.0 represents an attack with perfect attack and recall, while an AUC of even 0.6 may still be associated with strong performance (e.g. precision of 0.9 and recall of 0.7 would lead to an AUC of 0.63). In order to better understand why some websites are recognized more reliably than others, we

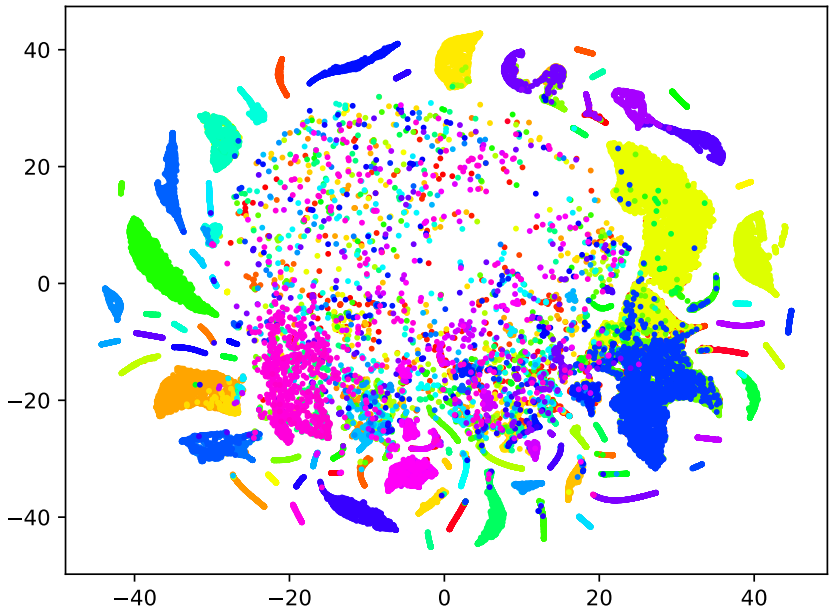

Figure 4.4: GTT23 test set embeddings, representing the top 100 classes, visualized with t-SNE. Note the mix of clearly defined classes, pairs of classes that are occasionally confused, and classes that are inconsistently embedded by the base model.

again create embeddings of the test set by taking the representations used by the attack in the second-to-last layer, then plotting them in two-dimensional space using t-distributed stochastic neighbor embedding (see Figure 4.4). We find that, while a substantial portion of the websites are associated with consistent and well-separated embeddings, there are sets of other websites that are frequently confused with one another. Furthermore, there is also a large subset of traces that do not appear to be recognized by the model, as they are embedded near traces of a variety of other websites.

As discussed earlier in the chapter as well as in "High Precision Open-World Website Fingerprinting" by Wang et al. [76], high precision is more critical than high recall for carrying out a WF attack, as attacks without high precision will likely experience too many false positives for the attack to be useful. Additionally, users may repeatedly visit the same website, meaning

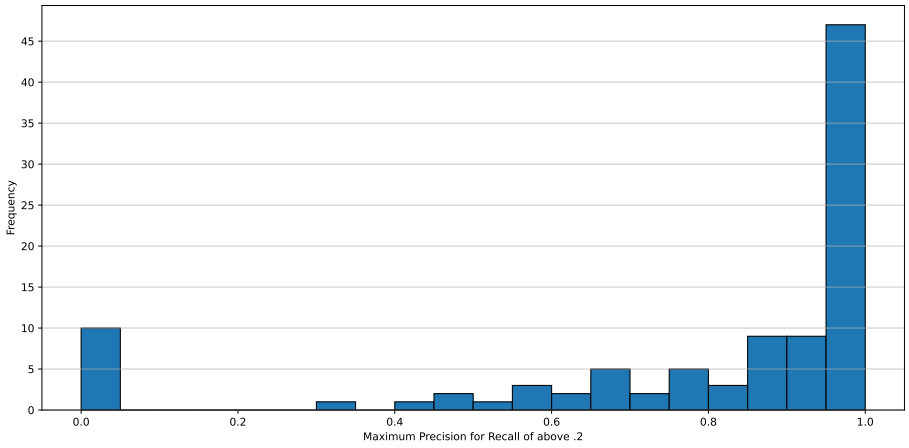

Figure 4.5: Per-class maximum precision for recall above 0.2 for the base attack

that a low-recall but high-precision attacker may eventually identify one of the visits.

Accordingly, we present in Figure 4.5 the highest precision achievable for a minimum recall of 0.2. This is computed by testing the precision and recall for varying attack output thresholds. While about 10 websites never achieve that level of recall for any precision value, and are thus assigned a value of 0.0, we find that 47 websites can be identified with a maximum precision of above 0.95, meaning that the attack can operate with a limited number of false positives for many of the websites. For the remaining websites, maximum precision tends to range from 0.4 to 0.95.

Optimized for Precision

To evaluate the *confidence* precision optimization technique, we again find the maximum precision associated with a recall of above 0.2. As shown in Figure 4.6, 10 websites again cannot be accurately identified, but the number of websites that can be identified with precision above 0.95 has increased from 47 to 67. This demonstrates that using the *confidence* technique to sacrifice precision for improved recall has worked for many of the websites. Accordingly, fewer of the websites have middling precision for recall above 0.2.

In Figure 4.7 we also provide the micro-average and macro-average precision and recall to compare the *confidence* technique to the other techniques. The micro-average precision-recall curve illustrates that high precision can be achieved for recall as high as 0.5 before starting to

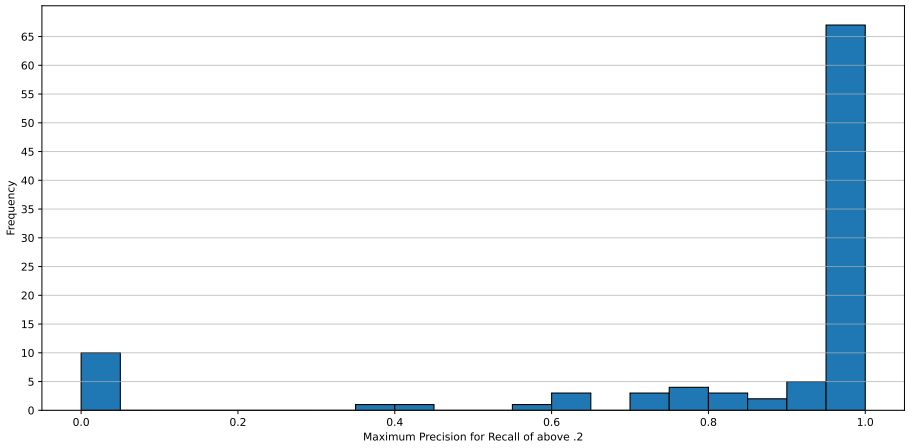

Figure 4.6: Per-class maximum precision for recall above 0.2 when using the *confidence* technique

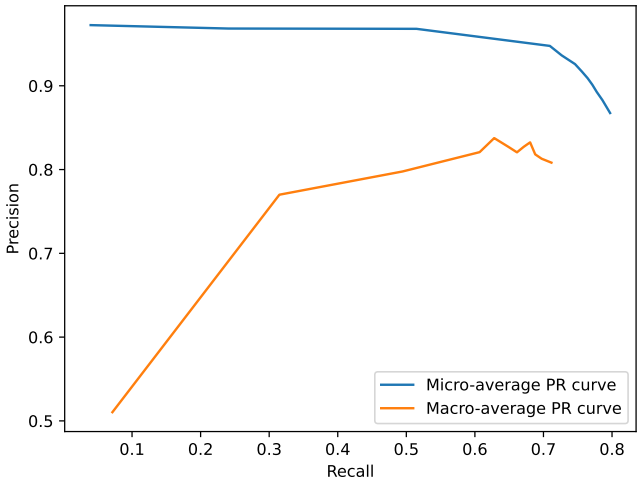

Figure 4.7: Micro-average and macro-average precision-recall curves for *confidence* technique

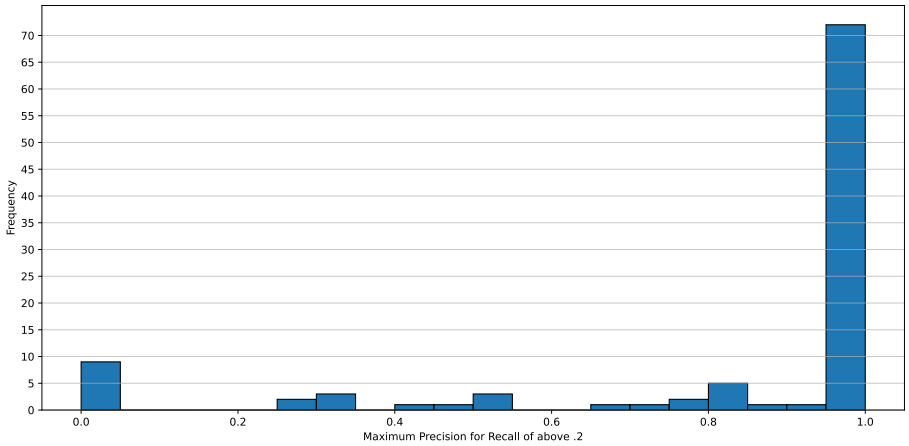

Figure 4.8: Per-class maximum precision for recall above 0.2 when using the *k-neighbors* technique

decrease. The macro-average precision-recall curve demonstrates substantially lower precision and recall, likely because a subset of the less frequently visited websites have very low precision and recall.

The maximum performance of the attack with the *k-neighbors* precision optimizer, shown in Figure 4.8, is even further improved compared to that of the *confidence* approach. In particular, there are only 9 websites that cannot be identified with appreciable precision with recall of above 0.2, while the number of websites associated with maximum precision of above 0.95 has increased to 72 from 47 with the base attack. As with the *confidence* approach, there are few websites associated with middling maximum precision.

The micro-average precision-recall curve, shown in Figure 4.9, again achieves a very high precision for a reasonable recall. Specifically, the maximum precision is as high as 0.996 for a recall of 0.48, indicating the high reliability of this approach. Again, the macro-average precision and recall are significantly lower due to the few websites that are very difficult to identify.

Illustrated in Figure 4.10, the *ensemble* technique has the same number of websites with a maximum precision of above 0.95 for recall of above 0.2. Along with the small number of websites with middling precision, this demonstrates that the *ensemble* approach is able to effectively determine which traces can be reliably identified. However, the *ensemble* approach also appears to have difficulty identifying traces from 16 of the websites. This increase in the

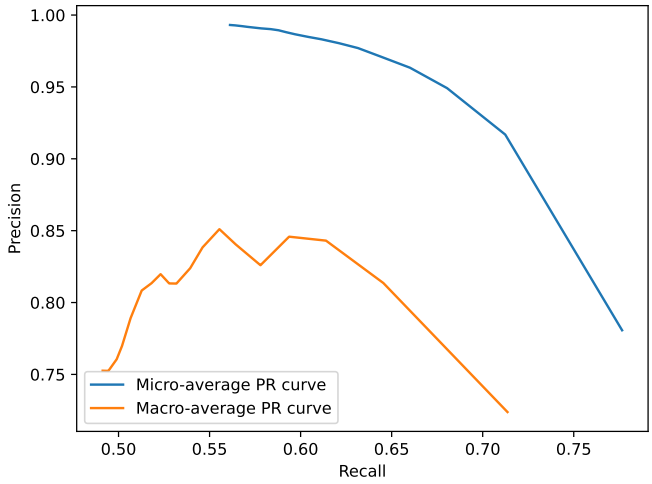

Figure 4.9: Micro-average and macro-average precision-recall curves for *k-neighbors* technique

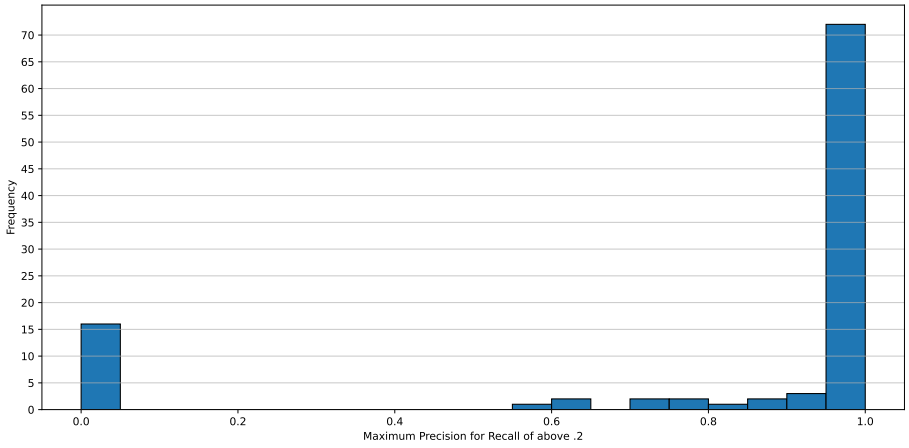

Figure 4.10: Per-class maximum precision for recall above 0.2 when using the *ensemble* technique

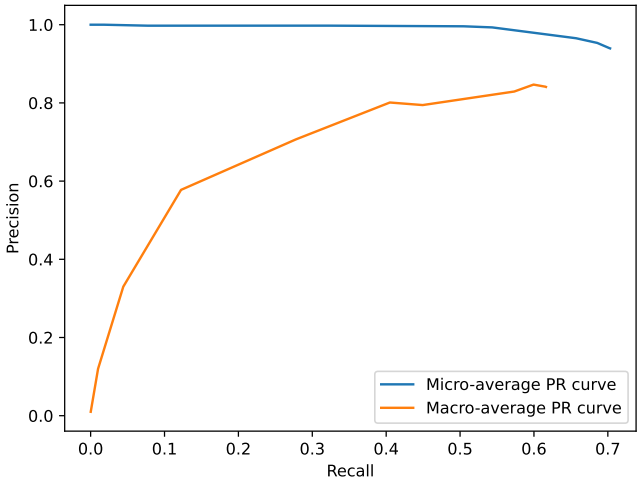

Figure 4.11: Micro-average and macro-average precision-recall curves for *ensemble* technique

number of websites associated with low attack performance is likely due to websites that one model, but not the other, can effectively identify with the given feature representations.

While the macro-average precision-recall curve again demonstrates the limited performance against some websites, the micro-average precision is very high for recall values as high as 0.5. In fact, for recall of 0.504, the associated precision is 0.996, meaning that about half of the monitored traces can be classified with very few false positives.

4.5 Summary

This chapter explores precision optimization techniques for WF attacks based on deep learning. Inspired by Wang et al.'s Precision Optimizers, our methods aim to enhance precision, which is critical for carrying out effective WF attacks in realistic settings. We implemented three precision optimization techniques - *confidence*, *k-neighbors*, and *ensemble* — and tested them against the Gong-Surakav and GTT23 datasets.

The *k-neighbors* and *ensemble* approaches perform exceptionally well on the undefended Gong-Surakav dataset, achieving 100% precision at 53% recall and 99% precision at 22% recall, respectively. Against defended datasets, the *ensemble* and *confidence* approaches perform best,

with *ensemble* approach achieving 78% precision at 6% recall against Surakav-heavy, for example. The *confidence* optimizer showed notable improvements against Tamaraw, achieving 85% precision at 45% recall. However, performance varies against defended datasets, with *k-neighbors* especially struggling due to obfuscated embeddings.

For the GTT23 dataset, the *k-neighbors* and *ensemble* approaches achieve the highest precision, identifying 72 websites with a precision above 0.95 for recall levels above 0.2. Both of these approaches then achieve a micro-average precision of 0.996 for recall around 0.5. Because the GTT23 dataset represents genuine Tor user behavior, these experiments provide strong evidence that WF attacks are indeed threatening in the real world.

Overall, precision optimization techniques effectively identify traces that can be reliably classified, allowing for higher precision at the cost of recall. This trade-off is advantageous for WF attacks, as minimizing false positives is crucial for practical, real-world applications.

Chapter 5

Improved Flow Correlation

In this chapter, we start with the Tor flow correlation attack DeepCoFFEA and adapt it for robust performance in both the flow correlation setting and for stepping stone intrusion detection (SSID). We call this adapted attack DeepCoFFEA-improved. Our additions include the use of multiple feature representations, neural network-based window combination techniques, a temporal drift representation, and other general training improvements. The motivation for these improvements stems from the high performance of DeepCoFFEA in Tor flow correlation, suggesting its potential effectiveness in other network security applications.

5.1 Related Work

5.1.1 Flow Correlation

Early flow correlation attacks focused on analyzing mix network traffic and used techniques ranging from 'packet counting' [45] to statistical correlation on binned packet volumes [51, 47] to mutual information analysis [52]. Later work on flow correlation attacks against Tor includes the RAPTOR approach, which uses Spearman's rank correlation coefficient to compare flows represented by their binned traffic volumes [53]. While RAPTOR achieves a degree of success, it often requires a long traffic flow to make reliable predictions. Furthermore, flow correlation on Tor is difficult to scale in the sense that a very low false positive rate is required to make precise predictions among the many potential flow observed flow pairings (e.g. N inflows and

N outflows have N^2 potential pairings).

Later, the DeepCorr approach [61] by Nasr et al. used deep learning to train a convolutional neural network to learn how Tor transforms traffic and determine which flows were correlated. They also collected a large and varying dataset, which helped the DeepCorr attack to make much more reliable predictions compared to RAPTOR. However, DeepCorr is computationally inefficient, requiring that the full convolutional neural network be evaluated for every pair of potentially matched flows.

To address these limitations, Oh et al. present DeepCoFFEA, which aims to perform flow correlation much more efficiently and with a lower false positive rate. Better computational efficiency is achieved by using a pair of convolutional neural networks to learn how to embed the inflows and outflows in a lower-dimensional space such that similar flows have similar embeddings. These models are trained using triplet loss, though DeepCoFFEA deviates from the usual triplet loss approach by training a separate model for the Tor inflows and outflows. To further reduce the false positive rate, the DeepCoFFEA approach splits the input traffic into temporal windows and provides separate embeddings for each window. Then, to make a final decision about which flows are correlated, the embeddings for each window are compared and determine to be matched if they are sufficiently similar. If the number of similar windows exceeds a set threshold, then the whole flows are predicted to be correlated. The voting process reduces the false positive rate, as it is very unlikely that uncorrelated flows are determined to be matched in most of the windows. As a result of these improvements, DeepCoFFEA functions effectively and relatively efficiently.

More recent work on flow correlation includes the Sliding Subset Sum attack [63], which is used to demonstrate an efficient flow correlation attack against Tor onion service sessions. The core of the approach is that the traffic flows are 'bucketized' into fixed time units, and a sliding subset sum algorithm is applied to compute similarity scores along a sliding window based on whether the sizes of some subset of packets sent by the onion service can sum to the amount of data received by the client. The highest scoring flow pairs, exceeding a set threshold, are then predicted to be correlated. While this approach appears effective and efficient for onion service flow correlation, we expect that the sliding subset sum method will not withstand traffic analysis defenses, such as 'dummy' packets and traffic delays, or changes in protocol. As a result, we do not consider adapting this approach for improved flow correlation or stepping stone intrusion

detection.

5.1.2 Stepping-Stone Intrusion Detection

Stepping-stone intrusion (SSI) is a method used by network intruders to obfuscate their origin and behavior while moving laterally across compromised hosts in the network. The connection chain can be constructed using SSH, HTTPS, or other bidirectional protocols. By increasing the length of the connection chain, the intruder becomes more difficult to track or expose.

To counter this type of threat, a variety of stepping-stone intrusion detection (SSID) approaches have been published. These typically take the form of either active SSID, which "watermarks" traffic, and passive SSID, which instead records and compares observed traffic flows. Here, we will only consider passive SSID, which we believe can be effective without requiring the network modifications and overhead required to alter network traffic.

Passive approaches are generally split into host-based SSID and chain-based SSID. Host-based SSID compares the flows going in and out of a given host and attempts to correlate flow pairs, thus determining whether that host is being used as part of a stepping stone. While host-based techniques have limited ability to distinguish between malicious and benign stepping-stone use, they can be used alongside chain-based SSID methods, which estimate the length of the connection chain and identify frequently malicious long chains.

Because DeepCoFFEA, described in the previous subsection, has demonstrated high performance in the Tor flow correlation setting, we aim to adapt it to function as a passive host-based SSID technique. Prior work on host-based SSID has taken a variety of forms, including correlating traffic connections that tend to send traffic at the same time [79], observing the long-term stream behavior while assuming a "maximum delay tolerance," [80], finding the deviation from one stream to another [81], comparing inter-packet delay statistics [82], and modeling traffic as a Poisson process to match packets between flows [83]. Additionally, an analysis by Blum et al. [84] presents basic detection algorithms while providing bounds on the number of packets needed to confidently detect an attack as well as the minimum volume of chaff required for an attacker to make streams appear independent.

5.2 Techniques

5.2.1 Multiple Feature Representations

While DeepCoFFEA originally used the series of packet sizes and inter-arrival times to represent each of the flows, we find that we can improve performance by using multiple feature representations to focus on various aspects of the input.[1] As described in Section 3.3, past work to quantify the amount of information leaked by different Tor traffic representations supports the possibility that this approach may increase the amount and usefulness of information available to the model [41]. For the flow correlation and stepping-stone intrusion detection settings, we achieve the highest performance while using a combination of the original packet size and inter-arrival times representations along with the directional time and cumulative representations, previously described in Table 3.1. In particular, the cumulative traffic representation, which represents the total amount of traffic sent or received at evenly-spaced points in time, significantly improves flow correlation performance against undefended Tor traffic. This is likely due to the high variance in traffic volume between flows, making total flow bandwidth a highly discerning feature.

Additionally, when handling the typically smaller volume traffic traces in the stepping-stone intrusion detection setting, we add an extra 'window' that contains information on all of the packets in the flow. This supports further comparisons of overall traffic volumes and longer-term traffic patterns in the compared traces.

5.2.2 Neural Network Window Combination

The original DeepCoFFEA flow correlation attack combines the similarities between the corresponding windows in the potential inflow-outflow pairs through a voting mechanicism; if enough of the windows are relatively similar, then the flows were considered matched. However, this approach is suboptimal when traffic traces vary significantly in terms of volume and time span, as some of the windows may have sparse traffic and thus be difficult to characterize. To address this, we investigated a variety of alternate methods to convert the window similarity values to a correlation prediction, including logistic regression, random decision forests, neural networks,

[1] The idea of using several feature representations for traffic analysis was developed in tandem with Nate Matthews. Several of the representations described in Chapter 3 were also developed together.

and gradient boosting. While all of these approaches performed relatively well, we find that the neural network approach led to the best flow correlation performance.

This neural network consists of three fully connected layers, two of which are hidden layers with dimensionality 128 and 32 respectively. The last layer then provides a single output which is interpreted, given a threshold, as a binary decision. Between the fully connected layers are rectified linear unit (ReLU) activation functions. To train the model, we use a validation set to generate window similarities between potential inflow-outflow pairs and then test whether the model correctly predicts whether the flows are correlated. Note that the training of this network is separate from the training of the base DeepCoFFEA models. We use the Adam optimizer with a learning rate of .0001 and binary cross-entropy loss.

By using this neural network architecture, we were able to capture complex relationships within the window similarity values that other methods might overlook. Illustrated by Figure 5.3, the neural network voting improvement alone accounts for most of the total improvement on the SSID-SSH dataset.

5.2.3 General Training Improvements

To best improve DeepCoFFEA performance and adapt it to the stepping-stone intrusion detection setting, we modify the deep learning approach to account for current best practices. This includes using learning rate warmup and cosine learning rate decay[2] to encourage fast and reliable convergence. Additionally, we increase the number of triplets sampled in each epoch before calculating the validation loss to reduce the potential of overfitting the model to the validation set. Last, we support multiple methods of selecting the negative examples for the triplet loss. While the original DeepCoFFEA approach uses the effective 'semi-hard negative mining' method, which balances between hard and easy examples [85], this approach is computationally intense due to its need to pre-compute distances between embeddings. As a result, we provide an option to randomly select negative examples during the training process, which is significantly faster. For example, when using an Nvidia RTX 3090 GPU and AMD Ryzen 9 5950x CPU, training time is 21ms per batch when randomly selecting negatives, compared to 85ms per batch when choosing semi-hard negatives.

[2]Using learning rate warmup and decay in this setting was inspired by code written by Nate Matthews

5.2.4 Temporal Drift Representation

With the addition of the neural network window combination technique, the DeepCoFFEA-based flow correlation process consists of embedding the windows of each flow, then computing the similarity of the embeddings for the respective windows for potentially correlated flow pairs. Then, these values are used by the neural network described in Section 5.2.2 to make the final decision regarding whether the overall flows are correlated. However, this approach may overlook valuable information regarding the temporal offset of the respective packets in the potentially matched flows. Consider that traffic entering Tor will likely exit the network very soon afterward, and that the recorded latency between entry and exit generally follows a reasonably predictable distribution. Furthermore, a potential flow pair that appears to illustrate traffic as exiting the Tor network before it enters can be excluded from further consideration. Accordingly, we find that it improves correlation performance to include the *temporal drift* between exiting packets and the nearest inflow packets. These values are simply used alongside the window similarity values in the second-stage neural network model, meaning that this approach is not significantly more computationally intensive than the base DeepCoFFEA approach. This addition does not substantially increase the training time for the second-stage neural network, which takes about 5.3 seconds per epoch on an Nvidia RTX 3090 GPU.

The temporal drift representation is based on the 'drift network' used by Oldenburg et al. [86] to perform flow matching against mixnets. Because Tor and other networks often obfuscate traffic enough to prevent accurate alignment of the packets in matched flows, we replace the approach of computing the sequence of time differences for all available packets with iterating through only the packets (or cells) in the outflow and computing the difference between it and the inflow packet that is nearest in time. Performance improvements when using the temporal drift approach are somewhat modest when performing Tor flow correlation, likely due to Tor's transformation of inflow traffic, but much more significant when doing stepping-stone intrusion detection.

5.3 Datasets

To test Tor flow correlation performance, we use the dataset originally collected to benchmark DeepCoFFEA [10] and refer to it as DCF. Its collection methodology is similar to that of the

DeepCorr approach [61], except that it collected additional traffic with a longer training-testing gap, avoided re-using circuits between the training and testing sets, collected traffic at the Ethernet layer, and used a smaller threshold to filter out small packets. The general collection methodology is to use a set of physical machines to act as Tor clients and crawl websites through a SOCKS proxy server. This way, the flows can be recorded both going into and coming out of the Tor network. Just over 40,000 flows were collected, though typically 20,000 are used in any given training run.

The SSID datasets were collected by Nate Matthews and other members of Prof. Matthew Wright's lab at RIT as part of an ongoing research collaboration. The collection process operates by sending traffic through a series of artificial stepping stones and then recording the packet size and timing information at each of the hops. The general traffic characteristics are based on that of public network traffic datasets, and various protocols are used to send the traffic. In particular, we use a dataset generated while using the SSH protocol, which we refer to as SSID-SSH, and another that uses both SSH and SOCAT, which we refer to as SSID-SOCAT. In the SSID-SOCAT dataset, some of the connections alternate between the protocols, making the correlation problem more challenging. For context, SSH (Secure SHell) is a protocol for secure remote login and data transfer, while SOCAT (SOcket CAT) is a tool for creating bidirectional data streams. The SSID-SSH datset contains 5,162 unique connections, while the SSID-SOCAT dataset contains 11,976.

5.4 Experiment Details

To test DeepCoFFEA-improved Tor flow correlation performance, our evaluation is similar to that of the original DeepCoFFEA paper [10], though we have re-implemented the preprocessing, training, and testing process in PyTorch [87]. We refrain from filtering out small packets when preprocessing the data and use 75% of the dataset for training and the rest for testing and validation. Like in the original evaluation, our training process uses the Adam optimizer and triplet loss with a threshold of 0.1. We also train the model for a high number of epochs, typically stopping training at 2,000 epochs and using the model associated with the best validation loss. Unlike in the original DeepCoFFEA evaluation, we lower the training loss to 10^{-4} and use learning rate warmup with cosine decay. As in the original implementation, we use a triplet loss margin of 0.1.

To train the second-stage neural network, which combines window and temporal drift information, we use the original validation set for model training and the test set for model testing. The precision and recall of the model are tested for varying thresholds, demonstrating how the approach can be tuned based on the desired reliability of the predictions.

The SSID experiments are similar, though they use only the first and last 'hops' of each connection in the datasets to benchmark performance. This best allows for the flows to be obfuscated while being sent through the simulated network, thus testing our approach in a challenging setting. Due to the short time span of many of the SSID flows, we typically 'shrink' the window sizes to best fit the flows; for the SSID-SSH dataset the windows are made 40x smaller, and for the SSID-SOCAT dataset they are made 2.5x smaller.

We train the models on an NVIDIA GeForce RTX 3090. Memory requirements are relatively low, requiring only about 2,500 MiB for training on the DCF-DeTorrent dataset with a batch size of 64. To simulate the DeTorrent defense on the DCF dataset, we use the configuration ($N_{download} = 3200$) described in Chapter 7, which leads to a bandwidth overhead of 97.3%.

5.5 Demonstration

5.5.1 Tor Flow Correlation

While DeepCoFFEA was originally intended for Tor flow correlation, and unsurprisingly has high performance on the DCF dataset, with a true positive rate (TPR) of 0.6 for a 10^{-5} false positive rate (FPR), the improvements listed above push the performance even higher. In particular, DeepCoFFEA-improved achieves a TPR of above 0.8 while empirically not having any false positives. It then manages to correlate nearly all of the flows for a FPR of 10^{-4}. This performance comparison is illustrated in Figure 5.1. In particular, adding the cumulative traffic representation appeared to greatly improve TPR for very low FPR levels. This improvement is particularly notable in that it demonstrates that flow correlation can be done against very large sets of potential inflows and outflows without the number of false positives making the technique unreliable.

Against DeTorrent, which is an adversarial zero-delay traffic analysis defense presented in Chapter 7, DeepCoFFEA-improved exhibited improved performance compared to DeepCoFFEA for all FPR levels above 2×10^{-5} (see Figure 5.2). For example, DeepCoFFEA-improved

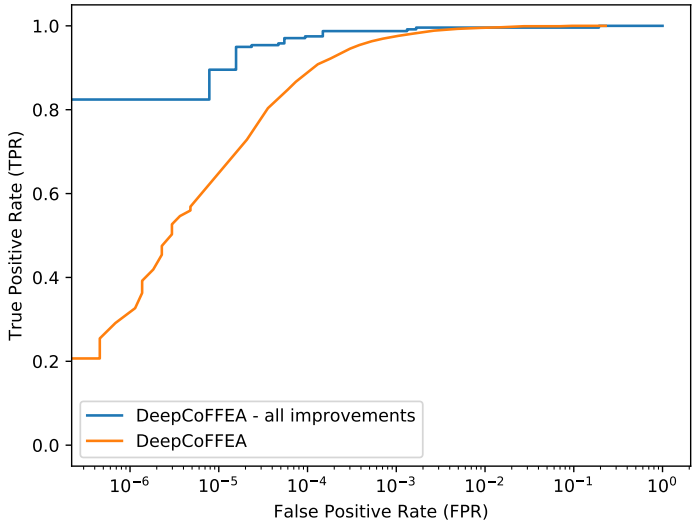

Figure 5.1: DeepCoFFEA vs DeepCoFFEA-improved on DCF dataset

TPR for a FPR of 10^{-5} was nearly 0.4 compared to less than 0.2 for the original DeepCoFFEA, demonstrating the impact of the improvements described above. However, both approaches have significantly reduced TPR compared to the results for the undefended dataset, with DeepCoFFEA and DeepCoFFEA-improved TPR reduced to less than half of their original values. So, while DeTorrent may not completely extinguish the threat of flow correlation, it does make the attack much more difficult to carry out successfully.

5.5.2 Stepping-Stone Intrusion Detection

The base DeepCoFFEA attack adapts fairly well to the SSID setting correlating flows with TPR of 0.6 for FPR of 10^{-3}. The initial improvement of the neural network voting method accounted for the bulk of the difference between DeepCoFFEA-improved and the base DeepCoFFEA, which suggests that the voting mechanism was suboptimal for SSH flow comparison (see Figure 5.3). This is likely due to the low traffic volume and frequently large gaps in the SSH flow data, which causes many of the windows in the inflows and outflows to be completely empty. However, the neural network voting can better make flow correlation determinations based on the similarity of just a few windows. When all flow correlation improvements are considered, the TPR rises

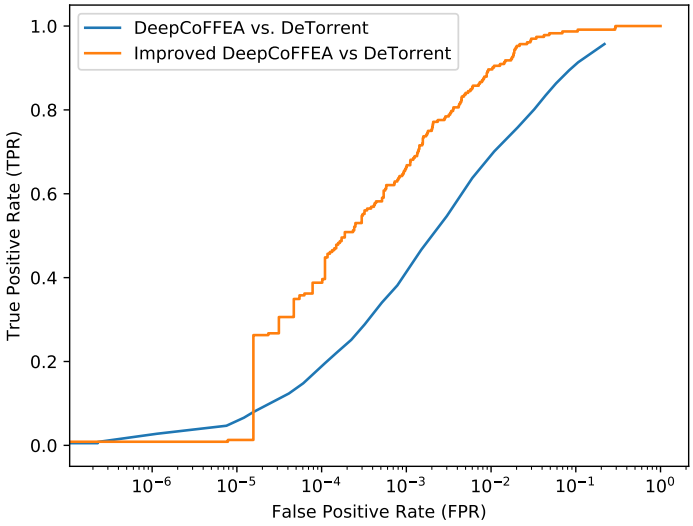

Figure 5.2: DeepCoFFEA vs. DeepCoFFEA-improved on DeTorrent-defended DCF dataset

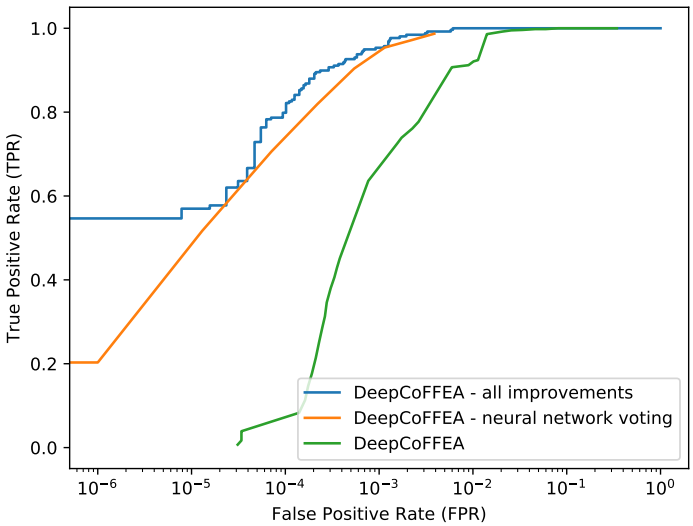

Figure 5.3: DeepCoFFEA vs DeepCoFFEA-improved on SSID-SSH dataset

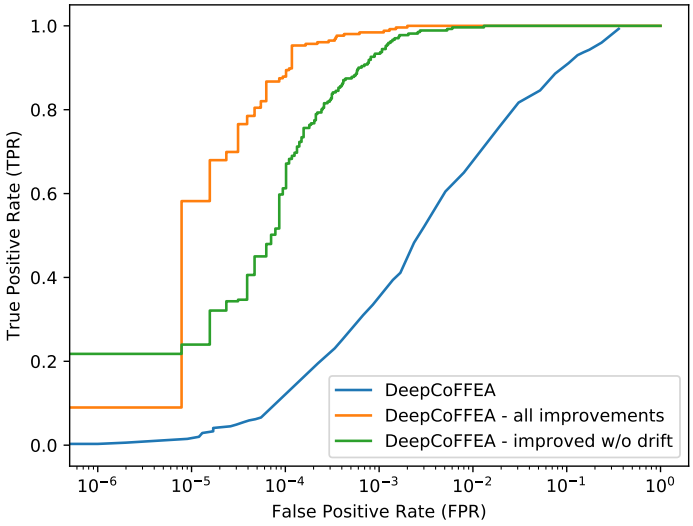

Figure 5.4: DeepCoFFEA vs. DeepCoFFEA-improved on SSID-SOCAT dataset

to above 0.55 for no false positives in our testing set. Then, for FPR of 10^{-3}, nearly all of the matched flows are being detected.

The base DeepCoFFEA model experiences worse performance against the SSID-SOCAT dataset compared to the SSID-SSH dataset for FPR levels greater than 10^{-3}. However, this trend is reversed for lower FPR levels. DeepCoFFEA-improved TPR at the 10^{-5} FPR level is around 0.6, which is similar to the results against the SSID-SSH dataset. However, in this case, the performance is most substantially improved with the temporal drift representation, more than doubling performance at 10^{-5} FPR, as shown in Figure 5.4. Again, the TPR approaches 1 as the FPR starts to become greater than 10^{-3}.

5.6 Summary

In this chapter, we present DeepCoFFEA-improved, an enhanced method for flow correlation and stepping stone intrusion detection. One key improvement is the use of multiple feature representations, combining packet size, inter-arrival times, directional time, and cumulative traffic representations to capture various aspects of the input traffic. Furthermore, the original

voting mechanism is replaced with a neural network approach to better handle variability in traffic volume and time span. Additionally, temporal drift between exiting packets and the nearest inflow packets is incorporated to improve correlation performance. We also apply current best practices in machine learning, such as learning rate warmup, cosine learning rate decay, and improved triplet sampling methods, to enhance the training process.

Our experiments demonstrated that DeepCoFFEA-improved achieves higher true positive rates and lower false positive rates in both Tor flow correlation and SSID settings. Specifically, the cumulative traffic representation significantly improved performance against undefended Tor traffic, and the neural network voting method proved to be effective for low-volume traffic in SSID. Additionally, the temporal drift representation provided substantial performance gains in SSID scenarios. These enhancements make DeepCoFFEA-improved a robust and efficient tool, capable of effectively identifying correlated flows and detecting intrusion attempts.

Chapter 6

RegulaTor: A Straightforward Website Fingerprinting Defense

WF attacks are straightforward to carry out, as they require only passive eavesdropping from a local adversary, ISP, or Tor guard. Thus, WF attacks against Tor users represent a realistic threat and may be used to identify users who visit 'censored' or forbidden websites. As this poses an obvious threat to users' privacy, researchers have developed a variety of defenses [28, 88, 13, 27, 30, 21, 31]. The goal of these defenses is to alter traffic in a manner that makes it difficult to determine which website is associated with each packet trace, and they typically operate by strategically adding 'dummy' packets or by delaying packets.

However, the Tor Project has been hesitant to implement past defenses, as most would either impact user experience with increased latency, burden the Tor network with increased bandwidth, or require the creation and maintenance of additional infrastructure. Additionally, many of these defenses have been proven ineffective against the latest attacks, which utilize large datasets and sophisticated deep learning techniques. In response, we present RegulaTor, which provides strong protection against state-of-the-art WF attacks with moderate bandwidth overhead and a small latency penalty, but without requiring additional infrastructure or knowledge of other traces.[1]

[1]This chapter is based on the paper "RegulaTor: A Straightforward Website Fingerprinting Defense" by Holland et al., published in the Proceedings of Privacy Enhancing Technologies 2022.

Our key observation is that defenses that "regularize" traffic so that traces from different web pages are identical tend to be the most effective. However, these defenses are often the least efficient, as they incur high latency overhead in periods of heavy traffic and high bandwidth overhead in periods of light traffic. Still, constant rate traffic is *just one of many* potential patterns for traffic regularization. Based on empirical evaluation of Tor web traffic, we find that there are common traffic patterns that avoid these traffic rate mismatches, allowing users to achieve the security benefits of regularization while greatly reducing the associated overhead.

Accordingly, RegulaTor works by regularizing the size and shape of packet 'surges' that frequently occur in download traffic, masking potentially revealing features. In this paper, 'surge' is broadly defined as a large number of packets sent over a short period of time. To do this, whenever a download traffic 'surge' arrives, RegulaTor starts sending packets at a set initial rate to avoid leaking information about the volume and length of the surge. Then, it decreases the packet sending based on a set 'decay rate' parameter, which defines the shape of the surge. If no packets are available when one is scheduled, a dummy packet is sent instead. However, due to the heavy 'burstiness' of web-browsing traffic, this download padding approach can be carried out with limited overhead. At the same time, RegulaTor sends upload packets at some fraction of the download packet sending rate. Moreover, sending upload packets based on download traffic usually incurs little latency overhead, as upload traffic mimics the download traffic (albeit with less volume) in web browsing traffic, as shown later in this paper.

The RegulaTor approach deviates significantly from previously proposed WF defenses. First, it alters traffic in a time-sensitive manner with a focus on sending standardized surges, while other defenses (with FRONT [31] a notable exception) tend to insert padding consistently along the packet sequence. Furthermore, it uses entirely different strategies to alter upload and download traffic, while other defenses pad traffic consistently regardless of direction. Lastly, RegulaTor uses the observed similarity between upload and download traffic to send upload traffic as a function of download traffic, preventing upload traffic from leaking any further information.

We also re-evaluate previously presented attacks and defenses on our dataset to enable direct comparison. In the closed-world setting with 95 websites, RegulaTor reduces the accuracy of the state-of-the-art attack, Tik-Tok, to only 25.4% compared to 66.0% for FRONT-2500. Furthermore, it requires a latency overhead of 6.6% and a bandwidth overhead of only 79.7%, while FRONT-2500 requires a bandwidth overhead of 119%. In the open-world setting, RegulaTor's

performance advantage is even more severe, reducing the F_1-score of a precision-tuned Tik-Tok attack to .135 compared to .625 for FRONT-2500. Thus, RegulaTor drastically outperforms comparable defenses in terms of efficiently defending traffic in the open-world setting.

6.1 Related Work

While a full overview of WF defenses is given in Chapter 2, a short summary of defenses basic on traffic *Regulation* defenses will help to put the RegulaTor defense in context. *Regulation* defenses aim to make WF attacks difficult or impossible by regulating the packet sending. The original regulation defense, BuFLO [25], was primarily created to demonstrate that such a WF defense was possible, albeit inefficient. BuFLO operates by sending packets at fixed intervals for a set length of time. If no packet is available at the set time, then a dummy one is sent instead. Later, CS-BuFLO [26] was presented as an improved version of BuFLO that adapts its transmission rate to reduce overhead and congestion. Another BuFLO variant, Tamaraw [27] reduced overhead while functioning as a "theoretically provable" BuFLO. While these defenses are resistant to WF attacks and require no additional infrastructure or information about other packet sequences, their latency and bandwidth penalties are too high for practical use.

Still, WTF-PAD [30] manages to efficiently regularize some aspects of the packet sequence by using an approach based on adaptive padding [89], which was developed to prevent end-to-end traffic analysis. It does this by filling long gaps in packet sequences whenever the gap between packets is larger than the time length sampled from a distribution of typical inter-arrival times. By reducing the amount of information leaked by each trace while imposing only a moderate bandwidth penalty, WTF-PAD is a strong practical defense; accordingly, WTF-PAD is re-evaluated for comparison in this chapter.

The only non-regulation defense simulated in this chapter is FRONT by Gong et al. [31]. However, since FRONT is reasonably effective and moderate-overhead, it provides an excellent comparison to RegulaTor.

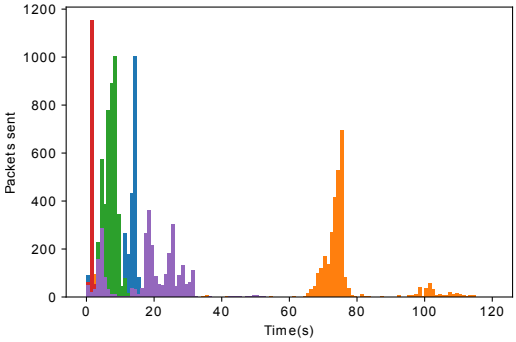

Figure 6.1: Examples of Tor download traffic during web page visits

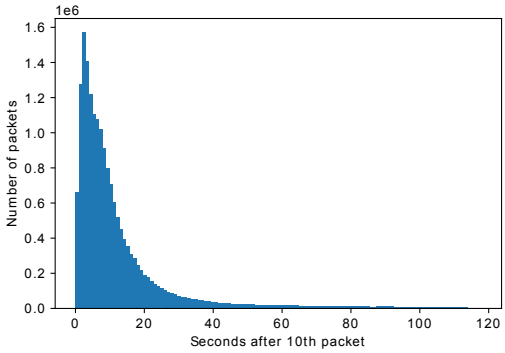

Figure 6.2: Decay of undefended download packet sending volume

6.2 RegulaTor

6.2.1 RegulaTor Justification

Given that the majority of relatively efficient previous WF defenses have been defeated, we aim to design a defense that can resist WF in both open-world and closed-world settings. We aim to keep the implementation simple and functional without requiring frequent tuning or collection of datasets, which is generally required in machine learning-based defenses. While this task is difficult, we find that there are common traffic patterns in Tor traffic that allow for an effective defense to operate with moderate overhead.

First, we analyzed DF-CW and found that the **packet traces are particularly surge-heavy** in that they consist of infrequent and irregular 'surges' of packets sent with little inter-packet delay. To illustrate this, we have plotted the download packet timing patterns from several randomly chosen traces (colored to distinguish between them) in Figure 6.1. Note that the traffic is characterized by occasional high-volume packet surges followed by periods of low traffic. We also find that, even though the average web page visit in DF-CW lasts about 28 seconds, the median interquartile range of packet times is only 3.96 seconds. This further demonstrates that the bulk of traffic is sent over a relatively short period of time. Furthermore, the location, size, and timing of these surges represent coarse features that leak a significant amount of information about the traces. To prevent these coarse features from leaking information about the traces, padding can be done in a more randomized manner to obfuscate the features, which was done in the FRONT defense; or, the surges in the packet sequences can be regularized in terms of size and location, which is the approach presented in this paper.

Luckily, the packet sequences show that **surge patterns are often predictable**. To be specific, a majority of the packet sequences consist of an early sequence of upload packets followed by a sudden surge of download packets. While the download surge varies in terms of size and start time, it generally decays in volume soon after the initial spike as the web page finishes loading. This is shown in Figure 6.2, which represents the packet distribution for 10,000 randomly chosen traces. In order to control for the start time of the first surge, only traffic after the 10th packet is shown. Additionally, the median packet is sent 7.57 seconds after the 10th packet, further emphasizing how the bulk of traffic is sent soon after the beginning of a web page visit. Accordingly, regularizing the download packet sequences can be done efficiently by adding dummy packets to sequences with smaller initial surges and delaying packets in sequences with larger initial surges.

Furthermore, the **timing of upload packets imitates the timing of download packets**, despite the relatively low volume of upload traffic, as outgoing requests are generally quickly responded to by the web page. The correlation between upload and download traffic volume is illustrated in Figure 6.3, where the traffic from 30 randomly chosen traces is split into 1-second bins and plotted based on the number of upload and download packets sent in that period. Most importantly, Figure 6.3 shows that if a substantial number of download packets are being sent, then upload packets are being sent concurrently. This presents an opportunity for RegulaTor

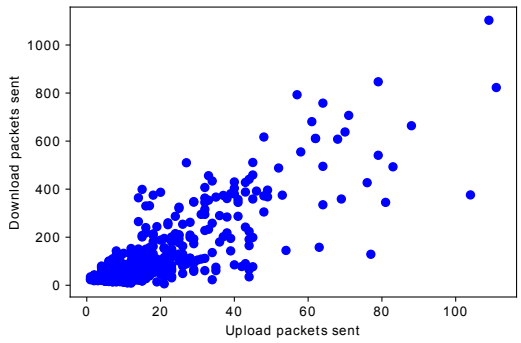

Figure 6.3: Download vs. upload traffic for each second

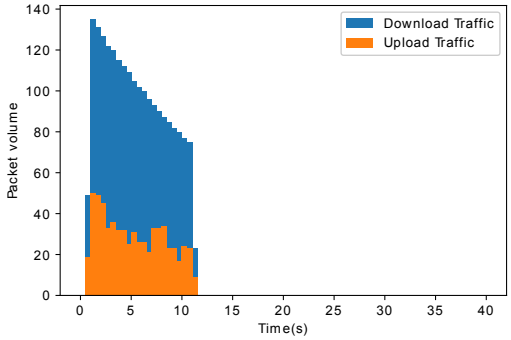

Figure 6.4: RegulaTor-defended trace

to defend the upload packet sequence as well: by modeling upload packet sending as a function of the download packet sequence, the upload traffic leaks no further information about the destination web page. Additionally, RegulaTor can minimize upload packet latency increases by intentionally overestimating the upload sending rate, which is relatively efficient given the lower volume of upload packets.

For a visual representation of the RegulaTor defense, see Figure 6.4, which illustrates the rate of download and upload packet sending at different points in time for a single defended web page visit.

6.2.2 RegulaTor Design

Parameters	Description
R	Initial surge rate (packets s^{-1})
D	Packet sending decay rate (s^{-1})
T	Surge threshold ratio
N	Padding budget (packets)
U	Download-upload packet ratio
C	Delay cap (s)

Table 6.1: RegulaTor parameter descriptions

To enact the defense, the client pads the upload packets, while the download padding can be carried out by a Tor bridge, middle node, or guard node. However, padding to the middle node is likely the most effective method, as a WF adversary may be located at the guard node. By only padding the traffic between the client and one of the first relays in the circuit, the real-world cost of the bandwidth overhead can be substantially minimized. In fact, Tor is generally bandwidth-limited by the relatively limited number of exit nodes [30], further minimizing the impact that RegulaTor-defended traffic will have on the network.

The RegulaTor defense pads the download packets as follows: the first download packets are sent at a constant rate until 10 packets have been sent. This is done to avoid sending the initial RegulaTor surge before the surge of original data has been scheduled while also allowing the circuit-building and TLS handshake to finish. At this point, RegulaTor begins to send a surge of packets at the initial surge rate, R, which represents the number of packets sent per second. However, the sending rate is reduced according to the decay constant, D, such that the sending rate is RD^t, where t is the number of seconds since the surge began. Afterward, if the original packet sequence again calls for a significant number of packets to be sent and the queue of waiting packets increases to some threshold, then RegulaTor sends another surge of download packets to minimize the potential for increased latency. This threshold is calculated as the surge threshold, T, multiplied by the target rate.

To minimize additional bandwidth overhead, the defense draws a random padding budget from $(0, N)$, where N is the maximum padding size. Once N dummy packets have been sent, the sending of dummy packets is stopped, though real packets may still be delayed. The random choice of N also functions to vary the total volume of the packet sequences, which reduces the

amount of information leaked by the volume of the sequence.

RegulaTor pads the upload trace at a constant rate until the initial download surge begins to arrive to accommodate the initial web page request. At this point, the client schedules upload packets to send at some ratio, U, of the download packet sending rate (e.g. the client will send upload packets at 1/3 the rate of the download packets for $U = 3$). U is chosen to minimize upload packet sending latency, so it will typically send a significant number of dummy packets; however, this is not costly bandwidth-wise, as the upload packet sequences are typically small. Additionally, if any upload packets have been delayed for more than C seconds, then they are sent immediately to prevent excessive latency.

Algorithm 1 demonstrates how the download packing sending schedule is determined, and table 6.1 contains the relevant parameters and their descriptions.

6.2.3 Parameter Tuning

To determine the specific parameter values for RegulaTor, we used the Tree-Structured Parzen Estimator (TPE) technique, which is usually used for hyper-parameter optimization for learning algorithms [90]. To implement parameter tuning, we used the Python library hyperopt [91] and determined RegulaTor's performance by simulating RegulaTor on DF-CW and testing the performance of Tik-Tok [8] against the RegulaTor-defended dataset. We chose Tik-Tok as the WF attack so that RegulaTor would be tuned to avoid leaking timing information, which Tik-Tok can detect with considerable effectiveness.

Then, to determine the performance of parameter combinations, we created a loss function based on a weighted combination of the latency overhead, bandwidth overhead, and WF attack accuracy. This also allows us to alter RegulaTor based on which properties (e.g. low latency) are most desired. In our evaluation, we present two RegulaTor defenses: one that achieves high performance at the expense of increased overhead (RegulaTor-Heavy), and another that aims to achieve moderate performance with lower overhead (RegulaTor-Light). Further details regarding the parameter tuning are available in the appendix.

6.2.4 Datasets

Our primary evaluation uses two datasets provided by Sirinam et al. that were originally collected in 2016 to test their Deep Fingerprinting attack [19, 92]. The closed-world dataset was

collected by visiting the homepages of the Alexa Top 100 sites 1,250 times each using tor-browser-crawler on ten low-end machines [93]. The homepage visits were split into five batches where for each batch, each machine would access a website 25 times before moving on to the next website. Batching the crawling in this manner controls for both long and short-term variance, as described by Wang et al. [15]. After discarding corrupted traces, 1000 instances of 95 sites were included in the final dataset, which we refer to as DF-CW.

The open-world dataset was collected using tor-browser-crawler [93] to visit the sites in the Alexa Top 50,000, excluding the top 100 sites crawled for the closed-world dataset. Again, ten low-end machines were used, with each machine making one visit to the home pages of 5000 different sites. After discarding corrupted visits, 40,716 traces were included in the open-world dataset, which we refer to as DF-OW.

Other datasets include the recently collected Goodenough dataset provided by Pulls [65] and the cell traces provided by Wang et al. [16] to demonstrate the k-nearest neighbors attack. Both are used to test RegulaTor's generalizability. The former dataset contains data for 100 websites with 90 instances each and was collected in 2014. The latter dataset, created in 2020, collected 20 samples from each of 10 web pages for 50 websites, resulting in a total of 10,000 samples. For both datasets, only the closed-world traces are used for straightforward evaluation and comparison between defense settings. In this paper, we refer to the k-nearest neighbors dataset as KNN and the Goodenough dataset as GE.

Furthermore, to test RegulaTor parameters in a real-world implementation, we use our pluggable transport implementation to collect 100 defended samples from each of the websites in the Alexa top 100. This dataset was collected over one month in August 2021 and is referred to as PT.

6.3 Defense Evaluation

In this section, we evaluate RegulaTor and several other WF defenses in terms of their closed-world performance, open-world performance, and overhead. Table 6.2 presents the parameters used in each of the examined defenses. For defenses other than RegulaTor, default parameters were used as provided by their authors [27, 30, 31].

For evaluation, we simulated each defense on the undefended datasets DF-CW and DF-OW

Defenses	Parameters
Tamaraw	$\rho_{out} = .04, \rho_{in} = .012, L = 100$
WTF-PAD	normal_rcv
FRONT-1700	$N_s = N_c = 1700, W_{min} = 1, W_{max} = 14$
FRONT-2500	$N_s = N_c = 2500, W_{min} = 1, W_{max} = 14$
RegulaTor-Light	$R = 260, D = .860, T = 3.75, N = 2080, U = 4.02, C = 2.08$
RegulaTor-Heavy	$R = 277, D = .940, T = 3.55, N = 3550, U = 3.95, C = 1.77$

Table 6.2: Defense Parameters

to create defended datasets for the closed-world and open-world settings. To simulate WTF-PAD and FRONT, we used the code provided by their authors. For Tamaraw, we used Tao Wang's implementation [94]. Then, we re-trained each WF attack on the defended datasets. The attacks used in this section used default parameters as well, except for CUMUL, which performs SVM hyperparameter tuning.

To demonstrate RegulaTor's generalizability, we also simulate RegulaTor on datasets used in past works, present the performance of a real-world RegulaTor implementation, and investigate parameter trade-offs. Then, we discuss parameter stability over time and the practicality of RegulaTor deployment.

6.3.1 Closed-World

Table 6.3 presents the accuracy achieved by WF attacks on the examined defenses. Since the experiment was done in the closed-world setting, accuracy is the only metric presented, and the DF-CW dataset was used. Each attack achieved high accuracy on the undefended dataset with Deep Fingerprinting achieving the highest accuracy. Tik-Tok and Deep Fingerprinting were highly effective against WTF-PAD, demonstrated moderate effectiveness against FRONT defenses, and were only marginally effective against RegulaTor defenses. However, CUMUL accuracy sharply decreased for all defenses. No attack was effective against Tamaraw, which is included as an example of a defense with strong theoretical foundations but impractically high overhead.

Defenses	Tik-Tok	DF	CUMUL
Undefended	97.0%	98.4%	97.2%
WTF-PAD	94.2%	92.4%	59.4%
FRONT-1700	78.2%	77.5%	31.6%
FRONT-2500	66.0%	69.8%	17.1%
RegulaTor-Light	34.8%	23.3%	20.8%
RegulaTor-Heavy	25.4%	19.6%	16.3%
Tamaraw	10.1%	9.9%	17.0%

Table 6.3: Closed-World Accuracy

Tik-Tok outperformed Deep Fingerprinting against the RegulaTor defense, likely due to the use of packet timing to provide further information. Given that RegulaTor generally sends upload packets at regular intervals, attacks that represent the traces using only packet direction, such as Deep Fingerprinting, are unable to achieve high accuracy. The usefulness of timing information was most apparent with RegulaTor-Light, which reduced Tik-Tok accuracy to 34.8% compared to 23.3% for Deep Fingerprinting. However, it should be noted that the RegulaTor parameters were tuned based on defense performance against Tik-Tok, while WTF-PAD and the FRONT defenses used default parameters. As a result, the RegulaTor defenses may have had a slight advantage against Tik-Tok.

Both RegulaTor-Heavy and RegulaTor-Light reduced Tik-Tok and Deep Fingerprinting accuracy significantly more than any other practical defense. Even when comparing the 'light' version of RegulaTor to the bandwidth-heavy version of FRONT, Tik-Tok accuracy is decreased from 66.0% to 34.8% and Deep Fingerprinting accuracy is decreased from 69.8% to 23.3%. However, CUMUL accuracy is not necessarily decreased when comparing FRONT to RegulaTor-Light. We suspect that this is because CUMUL derives a majority of its features from the cumulative representation of the trace, and FRONT effectively obfuscates early incoming and outgoing bursts in the trace, modifying the cumulative representation substantially.

Furthermore, RegulaTor-Light manages to outperform its rivals while incurring a small latency overhead and a bandwidth overhead similar to that of WTF-PAD. RegulaTor-Heavy then further increases this margin with a bandwidth overhead still less than that of the lower-overhead version of FRONT.

6.3.2 Open-World

Open-World Setup

While the closed-world setting is useful for illustrating the relative effectiveness of WF attacks and defenses, it is not a particularly strong indicator of real-world usefulness. As described earlier, the open-world model is characterized by a user who can visit any web page and an attacker who monitors a subset of those web pages while training a classifier to determine whether the visited web page is in that monitored set. This attack is typically more difficult to carry out, given that the attacker cannot train the model on many of the packet sequences in the unmonitored set.

In this experiment, when the attacker determines that a packet sequence represents a visit to a monitored web page, this prediction is a true positive if correct and a false positive otherwise. Similarly, if the attacker determines that a packet sequence represents a visit to an unmonitored web page, it is a true negative if correct and a false negative otherwise. An attacker is said to have determined that a web page is in the monitored set if the attacker's output probability is above a certain threshold. By varying this threshold, we can calibrate the attacks to achieve high recall or high precision.

However, true positive rate (TPR) and false positive rate (FPR) are not relevant measures of model performance, since the set of unmonitored web pages may be much larger than the set of monitored ones, causing heavily unbalanced classes. Furthermore, as the unmonitored class size grows, the number of false positives may begin to rival or surpass the number of true positives, making WF attacks impractical [3, 1]. Thus, as described in previous discussions of WF open-world metrics [3, 30, 18, 19, 31], we instead use precision-recall curves to evaluate WF attacks and defenses.

To carry out the open-world experiment, we use the two strongest WF attacks (Tik-Tok and Deep Fingerprinting) and evaluate them against the defenses used in the closed-world setting (except for Tamaraw, which prevented both attacks from reporting more than negligible true positives for many thresholds used). Our training and testing setup models that of the open-world experiment in Deep Fingerprinting [19], which used 85,500 monitored traces (900 each from 95 web pages) with 20,000 unmonitored traces in the training set and 9500 monitored traces (100 each from 95 web pages) with 20,000 unmonitored traces in the testing set.

Open-World Results

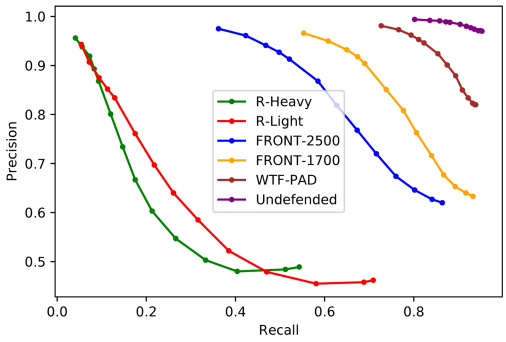

Figure 6.5: Tik-Tok precision-recall

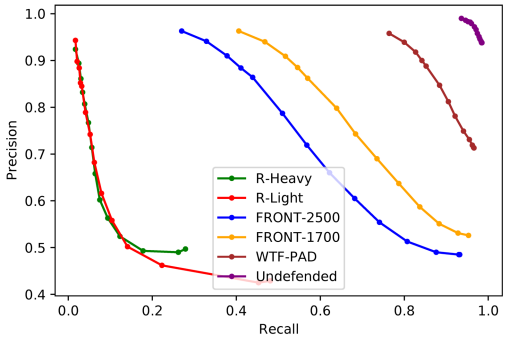

Figure 6.6: Deep Fingerprinting precision-recall

The precision-recall curves for the Tik-Tok attack against the WF Defenses are shown in Figure 6.5. As expected, Tik-Tok achieves both high precision and high recall in the undefended dataset, demonstrating the effectiveness of the attack in the open-world setting.

WTF-PAD and the FRONT defenses, alternatively, manage to drive down the precision-recall curve, forcing the attacker to choose between high precision with moderate recall and vice versa. However, FRONT-1700 does this more effectively than WTF-PAD, and FRONT-2500 further increases defense performance.

As shown, the RegulaTor defenses outperform their rivals, allowing for high precision only

with very low recall. As a result, the effectiveness of the Tik-Tok attack is significantly reduced, as few packet sequences can be determined to be in the monitored set with high reliability.

Against the open-world Deep Fingerprinting attack, the defenses are moderately more effective in terms of driving down precision and recall values, and the RegulaTor settings are about equal in terms of performance. The similarity between the RegulaTor defenses is likely the result of RegulaTor's ability to mask most features based on directional information. Again, RegulaTor shows significantly higher defense performance compared to FRONT, while both FRONT settings show improved performance over WTF-PAD.

To better compare the open-world performance of the tested defenses, consider that the F_1-scores of a precision-tuned Tik-Tok are .870, .625, and .135 against WTF-PAD, FRONT-2500, and RegulaTor-Heavy respectively. Note that a lower F_1-score implies that the associated defense is more effective. The relatively small F_1-score associated with RegulaTor further demonstrates its ability to prevent Tik-Tok from detecting a substantial number of monitored web pages without a high false positive rate as well.

Essentially, the open-world results indicate that the RegulaTor defenses prevent the studied WF attacks from achieving high degrees of precision and recall in a realistic setting. This is made even more apparent by the fact that the open-world setup used in this paper favored the attacker by using a test set where 9500 of the 29,500 traces were from the monitored class. In the real-world setting, it is unlikely that this proportion of all web pages would be monitored.

6.3.3 Alternate Datasets

To confirm that the RegulaTor defense generalizes beyond a given dataset, we test its performance on two other publicly available website fingerprinting datasets. For simplicity, we only evaluate Regulator-Heavy against the most effective WF attack (Tik-Tok) in the closed-world setting.

First, we test whether the parameters determined from tuning on one dataset provide effective defense for other datasets. To do this, we simulated the Regulator-Heavy defense setting on KNN and tested its performance against Tik-Tok. We found that it performed well, reducing Tik-Tok accuracy to 17.8% with 5.1% latency overhead and 77.3% bandwidth overhead. For comparison, Front-2500 defends KNN with a Tik-Tok accuracy of 44.9% and 98.3% bandwidth overhead. In both cases, slightly reduced accuracy is expected given that there are only 90 instances of each

website compared to 1000 in DF-CW, limiting the data available to Tik-Tok. Still, it appears that RegulaTor parameters can generalize across datasets, especially given that the datasets were collected a few years apart.

However, it is important to note that the ideal sizes of the packet surges in the RegulaTor defense are dependent on the volume of the undefended traffic sequence. If the RegulaTor surges are too small, then latency will unnecessarily increase as data waits to be sent. While DF-CW and KNN contained similar traffic volume, we may not be able to make this assumption in the real-world setting. To demonstrate this situation, we simulate RegulaTor-Heavy on GE, which contains much higher traffic volume at 5663.9 packets per trace compared to 2100.9 for DF-CW and 1807.6 for KNN. After simulating RegulaTor on GE, we find that RegulaTor-Heavy reduces Tik-Tok to an attack accuracy of 11.3% with 15.1% latency overhead and 39.6% bandwidth overhead. Here, the defense performance is high, but latency overhead is higher as well.

To adjust for the volume of traffic in the dataset, we can increase the initial surge rate and padding budget of the defense proportionally to the increased volume of the traffic in the target dataset. While this implies that some data collection should be occasionally done to implement the defense, this collection can be fairly minimal, as the adjustment only needs a rough estimate of relative traffic volume.

To demonstrate this adjustment using GE, we multiply the initial surge rate by 2.431, as this represents a relative increase of traffic volume in GE. Additionally, we increase the padding budget by a similar amount. This results in an altered Regulator-Heavy defense with an initial surge rate of 673 and a padding budget of 8030. When simulated on GE, Tik-Tok accuracy is only 5.2%, defense latency overhead is 2.9%, and bandwidth overhead is 82.9%. So, while bandwidth is increased, the altered defense is significantly more effective and operates with reduced latency overhead. For comparison, we test a FRONT defense with increased dummy packet volume to match the FRONT-2500 bandwidth overhead in the original paper [31]. Using a padding budget of 2830 on both the upload and download packets, we find that the FRONT defense reduces Tik-Tok accuracy to 43.4% with a bandwidth overhead of 45.8%. Thus, adjusted RegulaTor is still effective relative to FRONT.

In summary, RegulaTor-Heavy remains effective even when tuned on one dataset and then used on another. However, differences in traffic volume between the dataset used for tuning and the target dataset may cause RegulaTor to incur excess latency or bandwidth overhead.

As a result, the initial surge rate and padding budget should be proportionally adjusted as described previously. Then, RegulaTor-Heavy remains highly effective while incurring the intended bandwidth and latency overhead.

6.3.4 Real-world Performance

To obtain more accurate overhead estimates and confirm that RegulaTor could be smoothly used with Tor, we implemented the RegulaTor defense as a pluggable transport [95]. Pluggable Transports (PTs) transform traffic between the client and a bridge in order to disguise the Tor traffic and prevent censorship. Our specific approach was to host a Tor bridge and use the WFPadTools framework [96], which is based on the Obfsproxy pluggable transport [97], to build the RegulaTor defense. Additionally, we used a modified version of tor-browser-crawler [3, 8] to collect the traces for both the RegulaTor-Heavy defense and a 'dummy' transport for comparison. The parameters used in the pluggable transport version of RegulaTor were based on the RegulaTor-Heavy parameters, except that the initial surge rate and download-upload packet ratio parameters were adjusted based on the observed traffic patterns, as described in section 6.3.3.

Then, we used our pluggable transport implementation to collect a RegulaTor-defended dataset, PT, consisting of 100 websites and 100 samples per website. To determine the initial surge rate and padding budget, we first collected 10 samples from each of the websites and calculated that the average trace length was 2697.2. Comparing this to the traffic volume found in DF-CW, which was used to tune the original Regulator-Heavy, we then proportionally increased the initial surge rate to 356 and the padding budget to 4564. The remaining parameters were left unchanged, as they appear to generalize regardless of traffic volume.

Using Tik-Tok to test the closed-world WF attack on PT, we record an accuracy of 11.6%. Compared to the traces collected using a dummy pluggable transport, the adjusted RegulaTor-Heavy defense operates with a latency overhead of 13.9% and a bandwidth overhead of 78.2%. While the observed latency overhead appears somewhat higher than our original prediction (as discussed in section 6.3.5), the adjusted RegulaTor-Heavy defense is more effective as expected. These results demonstrate that RegulaTor can be effective on live traffic and that the parameters found from tuning on a previously collected dataset are still valid for a real-world implementation.

6.3.5 Overhead

Defenses	Latency OH	Bandwidth OH
Tamaraw	36.9%	196%
WTF-PAD	0%	54.0%
FRONT-1700	0%	81.0%
FRONT-2500	0%	119.0%
RegulaTor-Light	8.9%	48.3%
RegulaTor-Heavy	6.6%	79.7%

Table 6.4: Defense Overheads on DF-CW

Table 6.4 summarizes the bandwidth and estimated latency overheads for each of the tested defenses on DF-CW. The WTF-PAD and FRONT defenses operate with no additional latency, while the RegulaTor defenses incur a small delay on some packet sequences, and the Tamaraw defense causes a substantial delay. Still, since this paper measures latency overhead as the delay of the last 'real' packet, rather than the last dummy packet sent, Tamaraw's latency overhead may appear smaller than in previous works. Here, RegulaTor's latency overhead is an estimate based on the sum of the delay of the last real download packet and the maximum delay of any upload packet. The maximum delay of any upload packet is used because delayed upload requests may delay requests to the web server, delaying download packets even more so.

To illustrate how latency and bandwidth overhead are distributed across various websites, Figure 6.7 provides the bandwidth and latency multiples as functions of the original trace length, load time, and sending rate for 500 web sites randomly sampled from DF-CW.

As expected, nearly all 'short' traces and traces with a low sending rate have very low latency overhead. This is likely because RegulaTor schedules packets at a much higher rate than the original traces, delaying few packets. However, it is notable that packets with a very short load time often experience high latency overheads as well. This may be because traces with low loading times but high volume send packets at a high rate, which may exceed RegulaTor's sending rate. For the same reason, the sending rate of a trace appears to be correlated with latency overhead. Overall, a majority of traces incur little latency overhead, while traces with a high sending rate more often incur high latency overhead. Also, note that the distribution

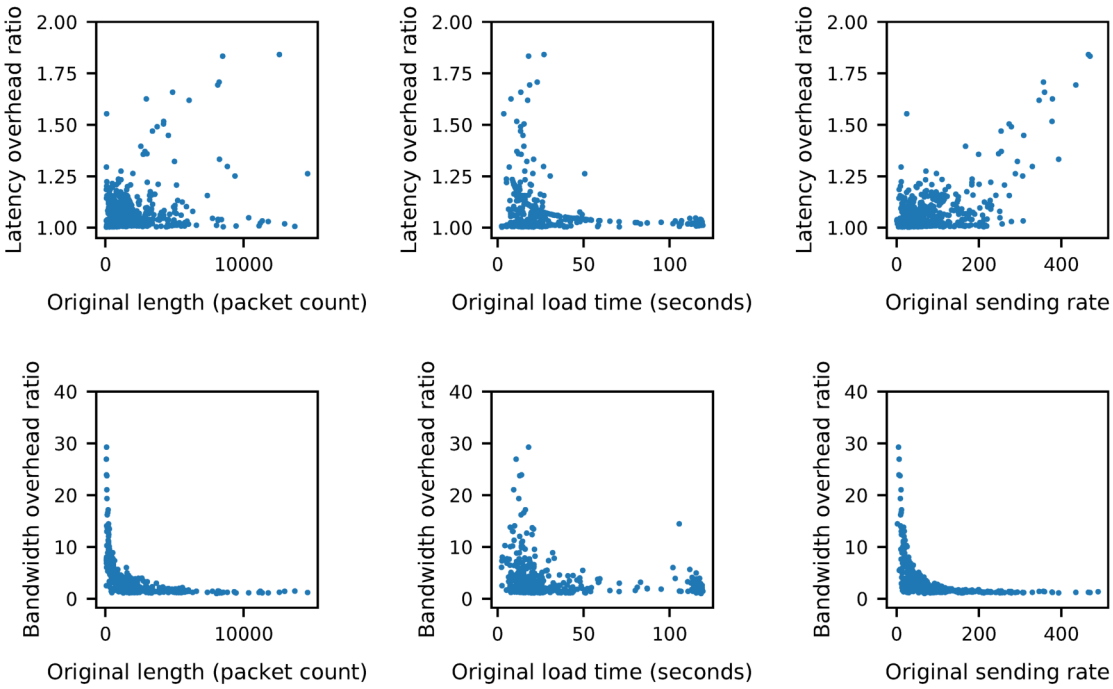

Figure 6.7: Overhead as a function of trace volume, load time, and packet sending rate

of latency overhead based on the web page being loaded is particularly important, as increased latency directly harms the user experience.

Additionally, low trace length and sending rate correspond with increased bandwidth overhead. This is expected, as RegulaTor will likely schedule packets at a much faster rate, which results in the frequent sending of dummy packets. Accordingly, traces with high sending rates are associated with low bandwidth overhead ratio. Also, a moderate number of traces with low load time incur high bandwidth overhead. This negative relationship appears to be because traces with low loading time tend to have smaller packet counts, with occasional exceptions.

Overhead-performance Trade-offs

To understand the impact of parameter choice, we also simulate RegulaTor on DF-CW with varied parameter values and record the latency overhead, bandwidth overhead, and Tik-Tok

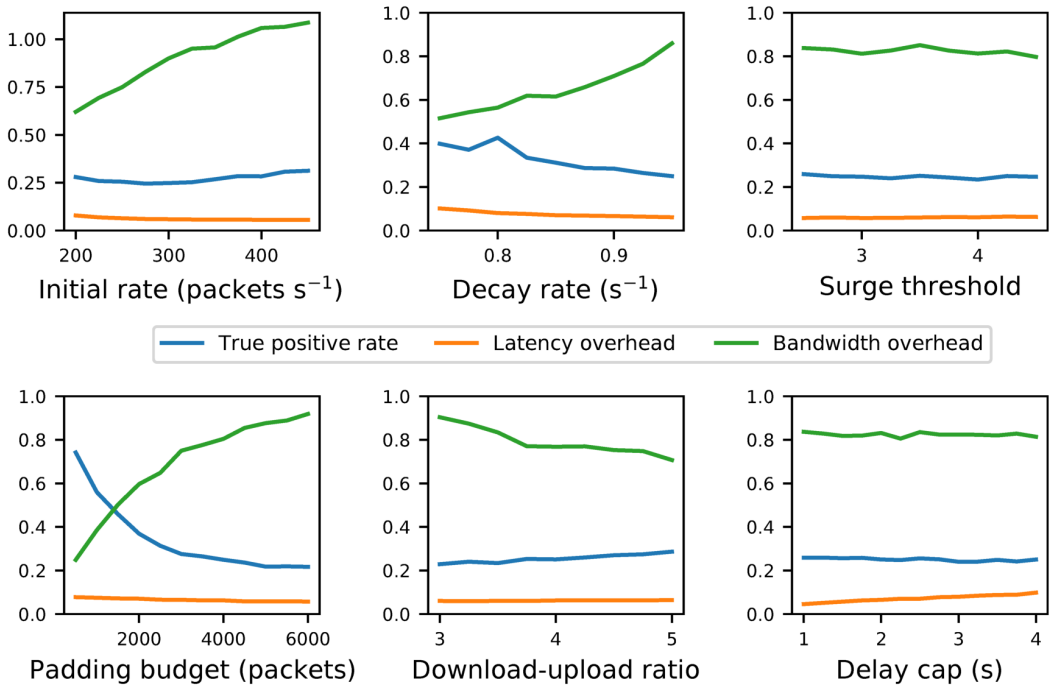

Figure 6.8: Overhead and closed-world performance by parameter value

performance on the defended dataset. To be specific, we used the RegulaTor-Heavy parameters and one by one varied the values of each parameter over a wide range. The results of these experiments are shown in Figure 6.8.

The experiments indicate that RegulaTor performance is fairly stable. Furthermore, varying the parameters allows for RegulaTor to choose different trade-offs between latency overhead, bandwidth overhead, and defense performance. Specifically, increasing the initial rate increases bandwidth and slightly decreases latency, while increasing the upload ratio decreases the bandwidth but weakens defense performance. Additionally, increasing the padding budget increases bandwidth overhead and defense performance while slightly decreasing latency overhead.

Varying burst threshold and delay cap parameter values had little effect on bandwidth overhead and defense performance, though increasing delay cap increased latency overhead. Lastly, increasing decay rate improved defense performance and decreased latency at the expense of increased bandwidth overhead.

Real-world Latency Overhead

While latency overhead can be estimated by simulating RegulaTor on Tor traces, the exact overhead is more difficult to determine, as one would have to accurately predict how the web server would respond to delayed upload packets. However, we can use the latency observed in the real-world dataset, which was collected using a PT RegulaTor implementation.

By comparing the loading time between the 'dummy' PT and the RegulaTor PT, we find that the average web page loading time for the RegulaTor defense increased 13.9%. While this exceeds the estimated latency for DF-CW, we find that this is partially explained by differences in traffic patterns.

Specifically, the standard deviation of load time recorded by our dummy PT, at 42.3, is substantially larger than that of DF-CW, which was 28.0. As a result, the real-world RegulaTor implementation more frequently encounters web pages with short load time and high volume or web pages with particularly long loading times. In the former case, RegulaTor will likely delay loading, as it will send packets slower than the high rate in the original trace. In the latter case, RegulaTor will have significantly slowed packet sending near the end of the trace, which may then cause increased latency if a late surge of packets is sent. Even with somewhat increased latency, RegulaTor defense performance in the real-world is strong, reducing Tik-Tok accuracy to 11.6%. If lower latency is desired, then parameters may be tuned to decrease it at the expense of a slight bandwidth or performance penalty.

6.3.6 Practicality

Due to RegulaTor's simplicity, it appears that the parameters do not have to be frequently retuned over time. Specifically, as long as web traffic can be characterized as predictably 'bursty' (as described in section 6.2.1), then the surge threshold, delay cap, and decay rate parameters will remain stable. This is further supported by RegulaTor's effectiveness on various datasets with these parameters unchanged. Furthermore, the download-upload packet ratio will remain stable as long as the real-world ratio of download to upload traffic remains constant. While this is difficult to predict, it is notable that both DF-CW and the data collected with the dummy PT, which are collected with similar methodologies but several years apart, report a download-upload packet ratio of 5.96 and 6.66 respectively. Thus, we expect that the download-upload packet ratio parameter will remain somewhat stable.

However, the initial surge rate and padding budget should be increased in response to changes in the average page load bandwidth. This increase appears likely to happen: according to the HTTP Archive [98], the average resource transfer size of tracked URLs increased from 1412.1 KB on July 1, 2017 to 2148.7 KB on July 1, 2021. Accordingly, the ideal initial surge rate and padding budget for the RegulaTor defense may change over time; fortunately, this change is likely to be predictable and straightforward to determine, as one only has to crawl a set of web pages and determine the change in average traffic volume.

In terms of the practicality of a real-world implementation, RegulaTor offers several advantages over previous defenses. First, RegulaTor's overhead is relatively moderate compared to existing regulation defenses, which tend to incur very high latency and bandwidth overhead. RegulaTor also operates without a database of traces associated with other websites, while defenses such as Glove [29], Supersequence [16], and Traffic Morphing [28] require knowledge of other traces. Lastly, RegulaTor does not require additional infrastructure or major changes to the Tor network, while application-level defenses such as HTTPOS [88] require browser modifications and traffic splitting defenses such as TrafficSliver-Net involve major Tor modifications. As a result of these advantages and RegulaTor's computational simplicity, RegulaTor can be straightforwardly implemented as a pluggable transport.

However, a pluggable transport implementation does introduce drawbacks. While PTs protect against the ISP and attackers on the client network, they do not protect against malicious bridges. Moreover, PTs require extra steps for users to set up and use, so many users are likely to remain unprotected. While a circuit padding framework exists [33], allowing developers to inject padding cells between the client and any node within the circuit, it does not allow for real cells to be delayed. Additionally, the circuit padding framework uses adaptive padding-style [30, 99] state machines to describe defenses. As a result, it is unable to support RegulaTor's approach of padding based on the timing of packet surges and buffering packets if necessary. Therefore, updating the circuit padding framework to support this style of defense is a remaining hurdle for the full deployment of RegulaTor.

6.4 Summary

This chapter introduces RegulaTor, a straightforward website fingerprinting (WF) defense aimed at mitigating the privacy threats posed by WF attacks on Tor users. WF attacks can be easily

conducted by passive eavesdroppers, such as ISPs or local adversaries, making them a realistic threat to user privacy. Traditional defenses often suffer from high latency, increased bandwidth usage, or the need for additional infrastructure, limiting their practicality. RegulaTor addresses these issues by providing strong protection against advanced WF attacks with moderate bandwidth overhead and minimal latency impact, all without requiring extra infrastructure or knowledge of other traces.

RegulaTor operates by regularizing the size and shape of packet surges in download traffic, masking identifiable features that could be exploited by WF attacks. It sends packets at a fixed initial rate during surges and then reduces the rate based on a decay parameter. This approach ensures that the packet surges are standardized, making it difficult to discern the website being accessed. Upload traffic is also padded proportionally to the download traffic, further obscuring traffic patterns.

The chapter demonstrates the effectiveness of RegulaTor through empirical evaluation, showing that it significantly reduces the accuracy of state-of-the-art WF attacks. In closed-world settings, RegulaTor reduces the accuracy of the Tik-Tok attack to 25.4% and the Deep Fingerprinting attack to 19.6%, compared to much higher accuracy rates for other defenses like FRONT. In open-world settings, RegulaTor further decreases the F_1-score of precision-tuned Tik-Tok attacks, showcasing its superior performance.

RegulaTor's design deviates from previous defenses by focusing on time-sensitive alterations and using different strategies for upload and download traffic. It also leverages the observed correlation between upload and download traffic volumes to minimize the information leakage. The chapter provides detailed descriptions of RegulaTor's parameter tuning, its implementation as a Tor pluggable transport, and its evaluation on various datasets, demonstrating its robustness and practical applicability.

Algorithm 1 RegulaTor download padding main loop

```
while < 10 packets scheduled do
    wait
end while

surge-time ← CURRENT-TIME
next-packet-time ← CURRENT-TIME
while web page downloading do
    if CURRENT-TIME ≥ next-packet-time then

        target-rate ← RD^(CURRENT-TIME−surge-time)

        if target-rate < 1 then
            target-rate ← 1
        end if

        if waiting-packets > T · target-rate then
            surge-time ← current-time
        end if

        if NUM-WAITING-PACKETS = 0 then
            if sent-dummy-packets < N then
                SEND-DUMMY-PACKET
                sent-dummy-packets ← sent-dummy-packets + 1
            end if
        else
            SEND-PACKET
        end if

        time-gap ← target-rate^−1
        next-packet-time ← next-packet-time + time-gap
    end if
end while
```

Chapter 7

DeTorrent: An Adversarial Padding-only Traffic Analysis Defense

Proposed traffic analysis defenses for Tor typically operate by either delaying traffic or by padding with 'dummy' traffic, obfuscating the relationship between the defended traffic and the associated flow. However, many defenses require either additional infrastructure or a prohibitive amount of latency or bandwidth overhead. While padding-only defenses may still incur some delay when widely deployed on Tor [100], we avoid explicitly delaying packets, as excessively increasing Tor latency is discouraged by Tor developers [33]. Accordingly, we aim to design an effective website fingerprinting and flow correlation defense that uses *only* dummy traffic.

The most recent and effective website fingerprinting and flow correlation attacks use deep neural networks (DNNs) to classify and link Tor traffic. DNNs are vulnerable to adversarial examples [101], which are intentionally crafted inputs that 'trick' a DNN into misclassifying that input. As a result, it may appear that adversarial techniques could be used to design a traffic analysis defense. However, adapting adversarial techniques for traffic analysis is not as straightforward as it may initially appear. First, the defense must operate while the traffic is being sent and without knowledge of future traffic, while the attacker can collect the relevant

data for later analysis. Second, the attacker will likely have access to the defense and can retrain the attack on a representative 'defended' dataset. In other words, the attacker model is not a static model targetable by adversarial perturbations; instead, it can adapt and retrain based on Tor's choice of traffic analysis defense. Thus, the defense must act in an unpredictable manner that resists adversarial retraining. Previous work on using adversarial methods for traffic analysis defense has achieved a degree of success [36, 34, 35]. However, these defenses either require knowledge of future traffic, struggle to prevent adversarial retraining, or delay traffic. Finally, the defense needs to be attack model-agnostic. For example, a defense model tailored to resist DNN-based attacks may not remain robust against alternate approaches, such as support vector machines.

To overcome these challenges, we present a novel approach, DeTorrent, to generate traffic padding strategies that defend Tor traffic.[1] Inspired by generative adversarial networks (GANs) [102], the core of the defense consists of a pair of competing neural networks that aim to either disguise or identify Tor traffic. More specifically, the 'generator' takes random noise as input and then schedules when and how many dummy packets should be sent. Then, the 'discriminator' (or attacker) attempts to demonstrate that it can accurately identify the defended Tor traffic. Because the generator's loss function is minimized when preventing the attacker from being able to identify the traffic, it is incentivized to generate effective padding strategies. Most importantly, those strategies are designed for resistance to adversarial retraining, as the generator and discriminator are trained in tandem.

Furthermore, to simulate the generator's real-time decision-making, we implement the generator as a long short-term memory (LSTM) neural network [103]. An LSTM is a type of recurrent neural network (RNN) that can be used to make repeated decisions while maintaining a state to store historical information. In the defense, the LSTM outputs how many dummy packets should be sent at each time step. Because the LSTM has feedback connections, it can generate this padding pattern while considering the previously monitored traffic. Once the generator is trained, it can be used to make effective real-time dummy padding insertion strategies.

To test the DeTorrent defense, we evaluate its ability to reduce the accuracy of state-of-the-art website fingerprinting and flow correlation attacks. We find that it outperforms comparable

[1] This chapter is based on the paper "DeTorrent: An Adversarial Padding-only Traffic Analysis Defense" by Holland et al., published in the Proceedings on Privacy Enhancing Technologies Symposium 2024.

defenses in both types of attacks while using similar bandwidth overhead. To illustrate, DeTorrent reduced closed-world Tik-Tok attack accuracy from 93.4% to 31.9%, which is 10.5% better than the next best defense. Additionally, DeTorrent reduces DeepCoFFEA's true positive rate to .12 for a 10^{-5} false positive rate, which is less than half of Decaf's TPR at .29. It does this while adding a bandwidth overhead of 98.9% in the WF setting and 97.3% in the FC setting.

To make our approach deployable, we provide a full implementation of the DeTorrent defense as a Tor pluggable transport and evaluate its performance against Tor live traffic. We further show the transferability of DeTorrent's padding strategies by training on one dataset partition and testing on another, leading to a minimal drop in performance of only .7%. Overall, DeTorrent demonstrates that a reasonable degree of traffic analysis defense can be achieved while only adding dummy traffic.

7.1 Related Work

A full overview of WF attacks and defenses is provided in Chapter 2. Still, there are a few other WF defenses that are especially relevant due to their use of 'adversarial' techniques. Mockingbird [34], for example, adapts adversarial example generation methods from computer vision, such as the Carlini and Wagner attack [104]. In particular, it modifies burst sequences to trick WF attackers. However, it also requires information about future traffic, which makes it difficult to implement. Another approach is the Blind Adversarial Perturbations (BAP) defense [35], which is where a model is trained to generate traffic perturbations compliant with traffic constraints. While it appears effective against static WF attackers, it does not defend against attackers that retrain their models to account for the defense.

Furthermore, Dolos [67] uses input-agnostic and location-agnostic adversarial patches to inject dummy packets into network traces. This approach appears to be effective with low overhead; however, it is tested against direction-only WF attacks, which appear to be considerably weaker than the best attacks in recent literature. Lastly, Surakav [36] employs a Generative Adversarial Network (GAN) to create realistic traffic traces that are used as reference patterns of packet sending. Surakav then dynamically adjusts these patterns while the web page is downloading. This approach also appears to be effective; still, since Surakav buffers traffic, it is not included for comparison against DeTorrent, which aims to not delay any packets.

7.2 DeTorrent Design

7.2.1 Motivation

A major difficulty faced by website fingerprinting defenses is that the attacker likely has access to the defense and defended traffic. Consequently, they can retrain their models or improve their approach to undermine the defense more effectively. To defeat this adversarial training, the most effective defenses have utilized the padding strategies based on a high degree of randomness as demonstrated by FRONT [31], Surakav [36], and Mockingbird [34].

As a result, it may seem intuitive to use an approach similar to that of a Generative Adversarial Network (GAN) to generate the defended traces and make adversarial retraining less effective. In this setup, the generator is fed random noise as input and trained to maximize the discriminator's loss. This way, the generator learns to insert dummy traffic in a manner that minimizes the discriminator's ability to identify the underlying traffic. However, this approach yields three challenges: representing the trace in a manner that can be used by neural networks, training the generator to make decisions about scheduling dummy traffic, and providing a notion of how well the discriminator can identify the trace. Our approaches for the FC and WF settings vary somewhat, so we now refer to the website fingerprinting version of DeTorrent as WF-DeTorrent and the flow correlation version as FC-DeTorrent.

7.2.2 Trace Representation

Representing the trace as a tensor is difficult given that the representations must support being 'added' to one another while serving as the output of one neural network and the input of another. This problem is further complicated by the fact that the LSTM generator must output the dummy padding pattern through a series of time steps. Also, because the generator's loss function is written in terms of minimizing the discriminator's performance, it must alter the traffic representation in a differentiable manner in order to use backpropagation to update its weights. In other words, we must be able to backpropagate through the discriminator's output and back to the generator's weights. This prevents us from representing the traffic as a series of packet directions or timestamps, as shown in existing WF attacks such as Deep Fingerprinting [19].

We find the best approach to account for these difficulties is to bin the packets of the trace

such that the value of each bin is the volume of download packets sent during a designated time period. This way, the traces can be 'defended' by simply adding the tensors together. This representation only considers download packets because the upload traffic is generally both smaller in volume and distributed similarly compared to the download traffic, as discussed in past work [105]. Thus, the upload padding is handled based on observed download traffic, as discussed in Section 7.2.5. Because a majority of traffic is sent soon after a web page is loaded, the bins are spaced evenly on a logarithmic scale using the NumPy function 'geomspace' [106], allowing for more fine-grained distinction between early traffic bursts. However, the bins are only fit to the data in the sense that they span 50 seconds, which is enough to contain the traffic in nearly all web page loads. The timing for each bin is also consistent for all traces. As a result, bin spacing will most likely not need to be re-tuned even if web browsing traffic patterns change.

While a binned representation loses some fine-grained information about the traces, this appears to be minimal compared to the amount of information leaked by the overall distribution and volume of traffic. We test this by running a closed-world WF classification attack on the DF dataset using only the binned representations and find that accuracy remains high at over 90%. This is further suggested by the success of WF attack CUMUL [18], which uses the cumulative distribution traces, and the relative success of WF defenses, such as FRONT, RegulaTor, and Surakav, that primarily alter the volume and distribution of traffic [31, 105, 36]. Furthermore, the success of the recent Robust Fingerprinting attack, which uses time slots, or bins, to create a 'robust' traffic representation, is further evidence of the usefulness of this representation [6].

Another potential limitation of the binning is that the DeTorrent generator is unable to learn the precise timing of dummy traffic and instead must rely on heuristic strategies, as described in Section 7.2.5. However, a more fine-grained version of DeTorrent would likely also struggle to respond to observed real traffic quickly enough to be effective, as this would require much more frequent LSTM evaluations. Having smaller bins that the beginning of traces also helps to mitigate this somewhat.

To preprocess the data, we first shift the traffic so that the binning starts with the 10th packet, as the time for the website to begin loading is variable. However, we can prevent this variability from impacting the trace representations by 'subtracting out' the time it takes for the circuit to be constructed and for the web page to begin loading. When DeTorrent is applied to

real traffic, it sends additional dummy traffic with inter-packet delays drawn from an exponential distribution with a mean delay of one-tenth of a second. This obfuscates the circuit setup and initial communication with the website while allowing the real traffic to be sent without delay.

7.2.3 Trace Embedding

A naive implementation of WF-DeTorrent would create loss functions based on the discriminator's ability to classify traces (or correlate flows) and the generator's ability to increase the discriminator's loss. However, this style of training does not lead to an effective defense in the WF setting. This is because the generator is incentivized to defend the traffic in a manner that does not resist retraining, as it is easy to move the trace representation just across the WF discriminator's decision boundary and cause it to misclassify. However, the discriminator can retrain to account for the generator's minor perturbations. To provide a better metric for discriminator and generator performance in the WF setting, we chose an approach used by DeepCoFFEA [10] and trained an embedder to map traces to a low-dimensional space where trace similarity is measured using Euclidean distance.

To train the embedder to provide a useful notion of distance between traces, we use triplet loss, which compares an 'anchor' (A) input of a given class to both a 'positive' (P) input of the same class and a 'negative' (N) input from another class. Triplet loss is minimized when the distance between the anchor and positive embeddings is at least α smaller than the distance between the anchor and negative embeddings, where α is a set slack value.

$$\mathcal{L}(A, P, N) = \max\left(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha, 0\right)$$

Once the embedder is trained, the discriminator guesses the embedding of the original traffic while the generator defends the traffic to prevent this. By making accurate guesses, the discriminator indicates that it has enough information to identify the traffic. As discussed later, we find that an embedding dimension of 256 works best when considering defense performance and computational overhead. In the flow correlation setting, the discriminator identifies

whether the flow pairs are matched or not. Because this is a binary classification, the problem of short-sighted optimization does not occur, so embedding is not needed.

7.2.4 Real-Time Decisions

While a WF or FC defense could pre-generate padding strategies, it would likely underperform compared to a similar defense that takes into account traffic already sent or received during page load. For example, a real-time defense might decide to extend packet bursts or add fake traffic to traces with low packet volume. Similarly, the generator cannot take the entire trace as input, as the real-world implementation lacks knowledge of traffic that has not yet been sent. To train a generator to make these types of decisions, we implement it as an LSTM where each bin in the trace representation is matched to a time step of the LSTM. As a result, the number of decisions the LSTM makes is tied to the length of the trace representation, though this representation is of variable length. This way, the recurrent nature of the LSTM can be used to make repeated decisions while taking into account past traffic and padding decisions.

Considering that each bin is associated with a set time period, the LSTM can take as input random noise and the number of real packets in the associated *previous* bin at each time step, then output the number of dummy packets that should be sent at the *next* time step. This process is shown in Figure 7.1. While defense performance would be higher given the possibility to consider the number of real packets in the current bin of the trace representation, this cannot be known until the end of the associated time period, so the LSTM may only consider previous bins. Once bin 256 has been reached, padding stops until enough download traffic is sent to signal the loading of a new web page. Then, the defense restarts. The trace representation already ignores the first 9 packets of a page load, as described in Section 7.2.2, so we set this threshold to 9. Because the generator does not determine the amount of traffic in the first bin, this value is determined by randomly choosing a value from a uniform distribution ranging from 0 to 10.

7.2.5 Packet Timing

Download padding

While the generator output specifies how many dummy packets to send in each interval, DeTorrent must still specify the exact timing. Web browsing traffic is often burst-heavy with occasional

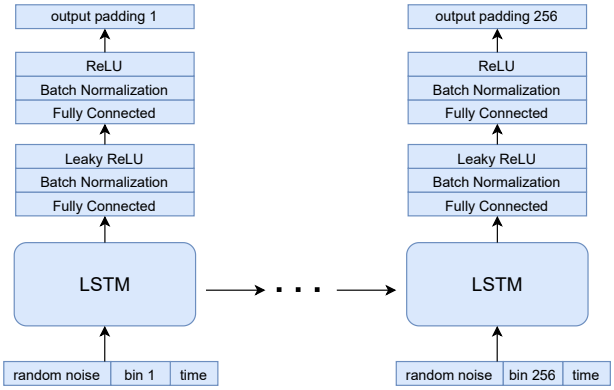

Figure 7.1: Unrolled LSTM generator outputting the dummy traffic volume at each step. Note that the volume in the previous bin, the time since the start of the defense, and random noise are used as LSTM input at each step.

long gaps between bursts [107, 108], so we choose inter-packet delays from the exponential distribution, as done in some previous works [109, 110]. The rate, λ, of the distribution is chosen such that the dummy traffic will on average fill the set bin (e.g. λ is set such that the average inter-packet delay will be 1/10th of a second given 10 dummy packets and a bin length of one second). The beginning of the burst is randomly offset according to a uniform distribution within the bin to prevent an attacker from identifying dummy traffic based on bin timing.

Upload padding

As mentioned earlier, Tor upload traffic tends to be distributed similarly in terms of when traffic bursts are sent, but at much lower volumes. As a result, DeTorrent's strategy is to simply add dummy traffic to maintain a set average of download-upload traffic. This achieves two goals: obfuscating much of the upload traffic and preventing the specific interleaving of the upload and download packets from leaking as much information. To be specific, DeTorrent sends dummy upload packets such that there is, on average, one upload packet for every five download packets. We chose this value for the parameter because, across all of the datasets used in this paper, few flows had a higher ratio of upload to download packets. This allows for DeTorrent to make nearly all the defended traces have the same download-packet ratio. To determine how many upload packets need to be added to achieve a given download-upload ratio, we must be able to create frequently updated estimates of the rate at which packets are sent or received. However,

this is challenging due to the irregularity of internet traffic and the need for the method to be computationally inexpensive. As a result, we implemented a traffic volume estimator that weights recently sent traffic more heavily, making the estimate highly responsive to traffic bursts.

Using the frequently updated estimates, DeTorrent sends an upload packet whenever the upload traffic rate falls below one-fifth of the download traffic rate. If the upload traffic rate is larger than one-fifth of the download traffic rate, then no dummy traffic is sent.

Estimating Traffic Rate

To determine the number of dummy upload packets to send so that the total upload rate is one-fifth of the download packet rate, we need to maintain accurate and frequently updated rate estimations for the rates of real upload and download traffic. However, this is nontrivial, as we'll need to find the rate of occurrence of an irregular event that is characterized by infrequent but large 'bursts' of activity. We also need responsive estimates in the sense that they quickly update to match the sending rate of traffic (rather than being lagging indicators) and we need to be able to compute an accurate average *between* packets. In other words, if an upload packet hasn't been sent in a while, then we should take that into account while computing an average. Lastly, for practical performance reasons, we would like to avoid searching for or storing information about previous traffic timing, as we'll have to compute the packet sending rate very frequently.

The approach described by Ilmari Karonen [111] first considers that we can model the true frequency over the whole time period as the integral of Dirac delta functions centered at the occurrence of the event divided by an integral of a constant (1) over time where λ_{packet_rate} represents the estimated packet sending rate:

$$\lambda_{packet\_rate} = \frac{\int \delta_{sent}(\tau) d\tau}{\int 1 d\tau}$$

However, we want an estimate of the *current* packet sending rate rather than the rate over the time period. So, we add a weighting function to emphasize the more recent packet sending:

$$\lambda_{recent\_rate} = \frac{\int \delta_{sent}(\tau) w(\tau) d\tau}{\int w(\tau) d\tau}$$

One reasonable choice of weighting function is to make it exponential, such as $w(\tau) = e^{k(\tau - t)}$ for a decay rate k. This also allows us to make the following simplification:

$$\lambda_{recent\_rate} = \frac{k\sum_{i=0}^{i=N} e^{k(t_i-t)}}{1-e^{k(t_0-t)}} \approx k\sum_{i=0}^{i=N} e^{k(t_i-t)}$$

for large $t - t_0$

To avoid having to save all of the event times, we can simply store the outcome of the most recent calculation and update it as follows:

$$\lambda_{recent\_rate}(t) = e^{k(t'-t)}\lambda_{recent}(t')$$

where t is the current time and t' is the time of the previous rate calculation.

When a new packet arrives, we'll account for it by updating as follows:

$$\lambda_{recent\_rate}(t) = k + e^{k(t'-t)}\lambda_{recent}(t')$$

Thus, we have a method of calculating the sending rate that can be tuned to be responsive to recent bursts, requires little information storage or computation, can handle the irregular nature of internet traffic, and allows us to accurately compute the rate both alongside and between the networking events. While the choice of k may vary, we find that $k = 1$ best fits the datasets used in this paper.

7.2.6 WF-DeTorrent Training

In the WF setting, the discriminator and embedder are both initialized as convolutional neural networks that take a trace representation as input and output an embedding. The training process, described in Algorithm 2 and illustrated in Figure 7.2, starts by training the embedder using the previously described triplet loss method with Euclidean distance. Then, after a random trace is sampled, the generator is fed a tensor containing random noise and the number of real packets sent at that time step. Next, the generator outputs the proportion of the padding budget, $N_{download}$, to be sent at that time step. Once the final dummy padding volumes are calculated, they are added to the original trace to create a defended trace. Using the defended trace as input, the discriminator outputs a prediction of the embedding of the *original* trace, indicating that it can accurately identify it. The discriminator's loss is computed as the distance

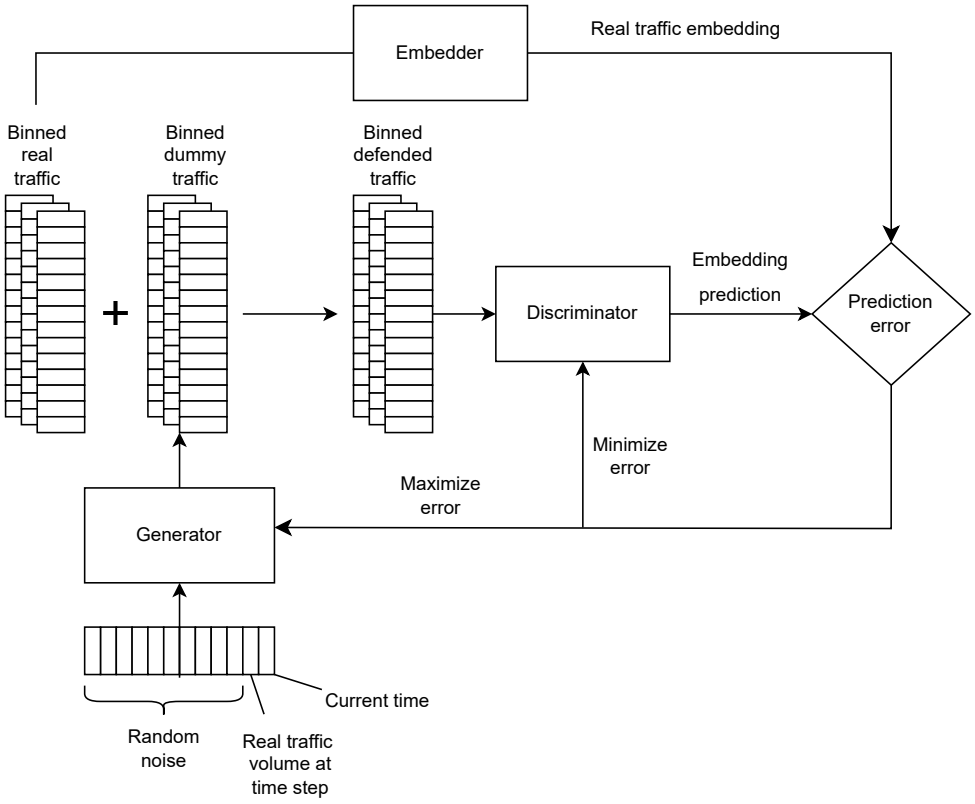

Figure 7.2: WF-DeTorrent training, where the generator minimizes the discriminator's ability to provide accurate embedding predictions. The binned real traffic and the binned dummy traffic are added to one another to create the defended trace, and the embeddings are computed using a pre-trained embedder.

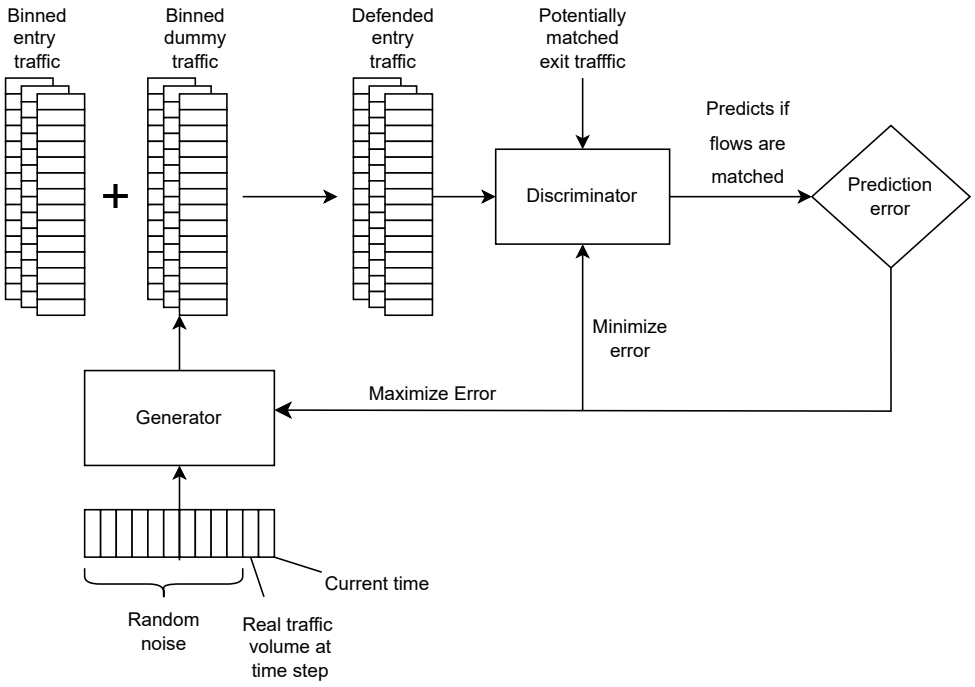

Figure 7.3: FC-DeTorrent Training, where the generator is trained to reduce the discriminator's ability to predict which flows are matched. The discriminator is trained to determine which flow pairs are associated with the same connection, and the generator is trained to defend the traffic to prevent this.

between its prediction and the true embedding of the original trace. So, given original trace x and noise z:

$$l_d(x, z) = -l_g(x, z) = ||D(x + G(z||x)) - E(x)||$$

Note that the generator only uses the traffic volume information of the current bin at each time step, and that '+' indicates a bin-wise addition of the traces. Also, we can use the $N_{download}$ value to tune the amount of overhead DeTorrent uses. In the training loop, once the loss is computed and the discriminator's weights are updated, the noise is re-sampled and the same trace is defended again by the generator. This time the process is similar, but the generator's loss is computed as the negative of the discriminator's loss. Re-sampling the noise and running the generator again appears to aid convergence, likely because it discourages the generator and discriminator from overfitting on the most recent set of noise vectors. We also experimented with training the discriminator or generator for multiple batches at a time but found that training each for one batch led to the highest defense performance.

We use the standard Gaussian to generate noise both because it is conventional with GANs and because the central limit theorem states that the sum of a large number of independent and identically distributed random variables is approximately normally distributed, regardless of the original distribution of the variables. Given the apparent universality to the Gaussian distribution, we hesitantly state that it is a reasonable default choice.

Note that the goal of the generator is to pad the traffic so that the discriminator is unable to provide an accurate embedding. While it might seem more natural to have the generator try to thwart a WF classifier, consider that accurately embedding a trace is equivalent to classification in the sense that this embedding could be used as input for a simple classifier. Accordingly, we find that this setup is an effective proxy for defending traffic to resist classification by a WF adversary and solves the traffic representation problem discussed in Section 7.2.3.

7.2.7 FC-DeTorrent Training

The setup for FC-DeTorrent is similar in the sense that it consists of an LSTM generator 'defending' traffic, which is then used as input to a CNN discriminator, as shown in Figure 7.3. However, here there is no embedder, as the generator directly minimizes the discriminator's ability to determine whether a given pair of flows is matched.

Algorithm 2 WF-DeTorrent Training

$E \leftarrow$ embedder model

$G \leftarrow$ generator model

$D \leftarrow$ discriminator model

$D_{train} \leftarrow$ training data

$\mathcal{L}_e \leftarrow$ embedder loss

$\mathcal{L}_g \leftarrow$ generator loss

$\mathcal{L}_d \leftarrow$ discriminator loss

$N_e \leftarrow$ number of epochs to train the embedder

$N_{adv} \leftarrow$ number of epochs to train the generator and discriminator

$m \leftarrow$ size of the generator input vector

for all epoch in N_e **do**

  for all batch b_i in D_{train} **do**

    sample anchors, positives from selected classes

    sample negatives from different classes

    anchor_emb $\leftarrow E$(anchor)

    positive_emb $\leftarrow E$(positive)

    negative_emb $\leftarrow E$(negative)

    $\mathcal{L}_e \leftarrow \frac{1}{|b_i|} \sum_{A,P,N \in b_i} max(||E(A) - E(P)||^2 - ||E(A) - E(N)||^2 + \alpha, 0)$

    update $E(x)$ weights to minimize $\mathcal{L}_e$

  end for

end for

for all epoch in N_{adv} **do**

  for all batch b_i in D_{train} **do**

    sample trace from D_{train}

    sample noise $z_0, ..., z_{m-1}$ from $\mathcal{N}(0, 1)$

    $\mathcal{L}_d = \frac{1}{|b_i|} \sum_{x \in b_i} ||D(x + G(z_0, ..., z_{m-1}||x)) - E(x)||$

    update D to minimize $\mathcal{L}_d$

    sample noise $z_0, ..., z_{m-1}$ from $\mathcal{N}(0, 1)$

    $\mathcal{L}_g \leftarrow -\frac{1}{|b_i|} \sum_{x \in b_i} ||D(x + G(z_0, ..., z_{m-1}||x)) - E(x)||$

    update G to minimize $\mathcal{L}_g$

  end for

end for

As described in Algorithm 3, the training process starts by sampling entry and exit pairs. 50% of the pairs are matched, as they may represent the entry and exit flows for the same traffic. Otherwise, they are unmatched. Then noise is sampled randomly and fed into the generator along with the number of real packets at each time step for the *entry* flow. Once the generator has output the proportion of the dummy download traffic to send at each time step and multiplied it by $N_{download}$, it is added to the original entry flow traffic. This traffic is used as input to the discriminator, which tries to predict whether the defended entry and original exit are matched. Note that the entry is defended because dummy traffic can only be sent between the client and a relay in the Tor network.

After making a batch of predictions, the discriminator's binary cross entropy loss is computed and its weights are updated. For entry flow e, exit flow x, true label y, and noise z:

$$l_d(e, x, y, z) = y \log D(e + G(z||e), x) + (1 - y) \log(1 - D(e + G(z||e), x))$$

Next, the noise is re-sampled and the generator again outputs a padding defense which is added to the real traffic to create a defended trace. The discriminator then takes the defended trace and original exit trace and predicts if they are matched. This time, however, the weights of the generator are updated to maximize the discriminator's loss, thus incentivizing strategies that make it more difficult to correlate the matched entry and exit pairs. Like in the WF setting, the generator only has access to the current traffic volume at each step, and the generator's output is scaled to achieve a specific bandwidth overhead.

7.3 Experiment Details

7.3.1 Datasets

The BigEnough dataset, referred to here as 'BE,' was collected for a Systemization of Knowledge paper on website fingerprinting defenses [68] by Matthews et al. The BE collection methodology was based on that of Pulls for the GoodEnough dataset [32] and contains three modes based on the Tor Browser security configurations: Standard, Safer, and Safest, though we only use the default standard mode in this paper. Most importantly, the BE dataset considers 95 websites represented by 10 subpages which are each crawled 20 times, resulting in 19,000 traces. Including

Algorithm 3 FC-DeTorrent Training

$G \leftarrow$ generator model

$D \leftarrow$ discriminator model

$D_{train} \leftarrow$ training data

$\mathcal{L}_g \leftarrow$ generator loss

$\mathcal{L}_d \leftarrow$ discriminator loss

$N_{adv} \leftarrow$ number of epochs to train the generator and discriminator

$m \leftarrow$ size of the generator input vector

for all epoch in N_{adv} **do**

  for all batch b_i in D_{train} **do**

    sample $entry, exit$ pairs from D_{train}

    sample noise $z_0, ..., z_{m-1}$ from $\mathcal{N}(0, 1)$

    $\mathcal{L}_d = \frac{1}{|b_i|} \sum_{entry,exit \in D_{train}} l_d(entry, exit, z_0, ..., z_{m-1})$

    Update D to minimize $\mathcal{L}_d$

    sample noise $z_0, ..., z_{m-1}$ from $\mathcal{N}(0, 1)$

    $\mathcal{L}_g = -\frac{1}{|b_i|} \sum_{entry,exit \in D_{train}} l_d(entry, exit, z_0, ..., z_{m-1})$

    update G to minimize $\mathcal{L}_g$

  end for

end for

subpages increases the realism of the dataset compared to past datasets, which model traffic as always visiting the home page of a given website. The Open PageRank Initiative [112] top website list was used to select the most popular websites. The BigEnough dataset also includes Tor log information needed to simulate the Spring and Interspace defenses for comparison; unlike other datasets, it also records the timing of Tor cells rather than that of the associated packets.

We use the website fingerprinting dataset by Sirinam et al., originally collected to demonstrate the Deep Fingerprinting attack [19], to evaluate defense performance against an attacker able to compile a large and robust dataset. The dataset was collected by visiting the home pages of the Alexa top 100 websites 1,250 times each using ten local machines. The crawl was done in batches to account for both long and short-term variance following the methodology described by Wang et al. [15]. After filtering, we are left with a dataset with 1000 instances of 95 websites, which we refer to as 'DF.'

To test the flow correlation defense performance, we use the dataset collected by Oh et al., referred to here as 'DCF,' to demonstrate the effectiveness of the DeepCoFFEA attack [10]. Their dataset collection methodology was based on that of the DeepCorr technique by Nasr et al. [61], though with some changes. Specifically, they used a web crawler along with a physical machine to collect the entry flows and a proxy server to collect the exit flows. Over 60,000 Alexa websites were crawled in over 1,000 batches with the first 60 seconds of each flow being recorded. The DCF dataset removes flows with less than 70 packets total or with a window packet count of less than 10. While the original dataset includes 42,489 flow pairs, we randomly selected 20,000 in our experiments.

To evaluate the DeTorrent pluggable transport defense, we visited each website in the Alexa top 250 [113] 100 times over two weeks in September 2022. To collect the dataset, we used a fork of Tor Browser Crawler [93] updated to work with more recent libraries and Tor Browser version 11.0.15. The crawler then uses the DeTorrent pluggable transport, described in Section 7.4.3, to transform traffic between it and the bridge. We ran two separate instances of the crawler, each in its own virtual machine, and connected to one of two Tor bridges. We hosted the bridges using Amazon Lightsail and used Tor version 0.4.7.8. In this paper, we refer to this dataset as 'PT.'

7.3.2 WF Attack Details

We used open-source code associated with WTF-PAD [114], Interspace [115], Spring [115], FRONT [116], and BAP [117] to generate the defended datasets for each defense. Then, to test the robustness of WF-DeTorrent and comparable defenses, we test the defense's ability to resist state-of-the-art attacks. For the website fingerprinting setting, we use the Tik-Tok and Deep Fingerprinting attacks. These attacks are based on the same CNN architecture, though Tik-Tok uses packet timing information, allowing it to achieve somewhat higher accuracy. We use the default parameters with the exception that we run the Tik-Tok and Deep Fingerprinting attacks for 100 epochs with a patience value of 10. This ensures that training does not stop early on the BE dataset, which is much smaller than the DF dataset and thus has much smaller epochs. We also increased the attack input size from 5,000 to 10,000 to account for the increased traffic volume observed in the BE, DCF, and PT datasets (compared to the DF dataset). For both Tik-Tok and Deep Fingerprinting, we use 5-fold cross-validation to estimate attack performance.

To fairly compare WF-DeTorrent to defenses such as FRONT and WTF-PAD, we scale the bandwidth overhead to match the overhead used by WF-DeTorrent. To scale FRONT, we increase the upload and download volume parameters N_s and N_c. To scale WTF-PAD using the distributions released in the original paper, we scale the timestamps of the datasets, use the WTF-PAD simulator to insert the dummy padding, and then re-scale the timestamps to match the original trace lengths. This strategy is used in past literature [32] and is necessary due to the smaller inter-packet delays seen in more recently collected datasets. To simulate BAP, we train the timing and direction perturbation attacks against Var-CNN as indicated by the source code shared by the authors. We also set the α parameter to tune the volume of the perturbations. The specific overheads and defense parameters are shown in Table 7.1 and Table 7.2. Also, it should be noted that a low but measurable amount of added latency may be incurred by a full deployment of these 'zero-delay' defenses, as shown in past work [100]. However, the exact amount of additional latency is difficult to estimate without a larger-scale Tor simulation.

The default Spring and Interspace defenses use a comparable amount of bandwidth overhead. Because the output of the circuit padding simulator [115] contains unrealistically precise timing information, we round the timestamps of the defended dataset to the nearest hundredth of a second. We only simulate Spring and Interspace on the BigEnough dataset, as their simulator requires Tor logs that are not present in the DF dataset.

To simulate WF-DeTorrent on a dataset, we must take steps to prevent it from being able to defend traces that it witnesses during training: this might allow it to memorize effective defense strategies and perform unrealistically well. So, we partition each dataset n-ways, where $n-2$ partitions are used for training, one is used for validation, and the other is used to simulate WF-DeTorrent and output the defended traces. Then, we rotate which partitions are used for training, evaluation, and output so that each partition is a test set exactly once, creating the defended dataset. While we do scale $N_{download}$ based on the choice of dataset, we do not do extensive hyperparameter optimization to tune DeTorrent, as it takes several hours to train in this manner.

To make clear comparisons between the website fingerprinting defenses, we test them in the closed-world setting where the attacker simply needs to determine which website a trace is associated with. While this setting is less realistic than the open-world setting, we prefer to test the defenses in a setting more advantageous to the attacker, as we can be certain that defense performance is not likely due to the experimental design. Furthermore, we make assumptions about the adversary's ability. These include that the adversary can determine the beginning and end of the page load, that the user visits the home page of each website, and that the adversary can create a dataset that accurately represents the target's unique network conditions. Again, these assumptions favor the attacker, so we expect real-world website fingerprinting to be at least as difficult as our results would indicate.

7.3.3 FC Attack Details

In the flow correlation setting, we use the DCF dataset and the DeepCoFFEA attack with default parameters to test the defenses. 10,000 flows from the dataset are used for training and 10,000 are used for testing. We train DeepCoFFEA for 1,500 epochs, as this tends to lead to the highest attack performance.

To simulate BAP in the FC setting, we train it to generate perturbations using the DeepCorr attack provided in the open-source BAP implementation [35]. We also set the α parameter to determine the volume of the traffic perturbations. For FRONT, we set the N_s and N_c parameters to achieve a comparable level of bandwidth overhead. To set the WTF-PAD overhead, we scale the timestamps of the DCF dataset, use the normal_rcv distributions as specified in the original paper, and then re-scale the timestamps to match the original. The specific parameter and

bandwidth overhead value for the defenses are shown in Table 7.5. We don't use the Spring or Interspace defenses in the FC setting, as they require Tor log information that the DCF dataset does not contain. To implement the Decaf defense, we randomly selected windows of traffic from dummy traces in the DeepCorr dataset [61] and used them to pad the real traffic traces, as described in the original 'Decaf-DC' implementation. To roughly match the volume of dummy traffic used by other defenses, we choose to apply the defense to all of the real traffic windows, rather than randomly selecting them.

To prevent FC-DeTorrent from having the unfair advantage of being able to train the generator and discriminator on the same dataset that it defends, we use the dataset partitioning method described in Section 7.3.2. We again refrain from doing hyperparameter optimization to tune FC-DeTorrent's training and implementation.

7.4 Implementation Details

7.4.1 Model Architecture

The discriminator and embedder in the WF setting use the architecture shown in Figure 7.4.1. This CNN architecture is based on the Deep Fingerprinting architecture with alterations so that it uses a tensor of size 256 as input and output. We also find that model performance is improved with Leaky ReLU rather than ReLU activation.

The discriminator in the FC setting is shown in Figure 7.4.1. Note that the outputs after initially processing the two flows are concatenated and then passed to the final layers. The output of the network is a prediction about whether the two flows are correlated where an output greater than .5 predicts correlation.

Note that the input of the generator, shown in Figure 7.1, is the concatenation of random noise and the volume of real traffic binned *at that time step*, and that the hidden state at each time step is used as the input to the fully connected layer. The generator LSTM has a hidden size of 128, an input size of 32, and is made up of a single layer.

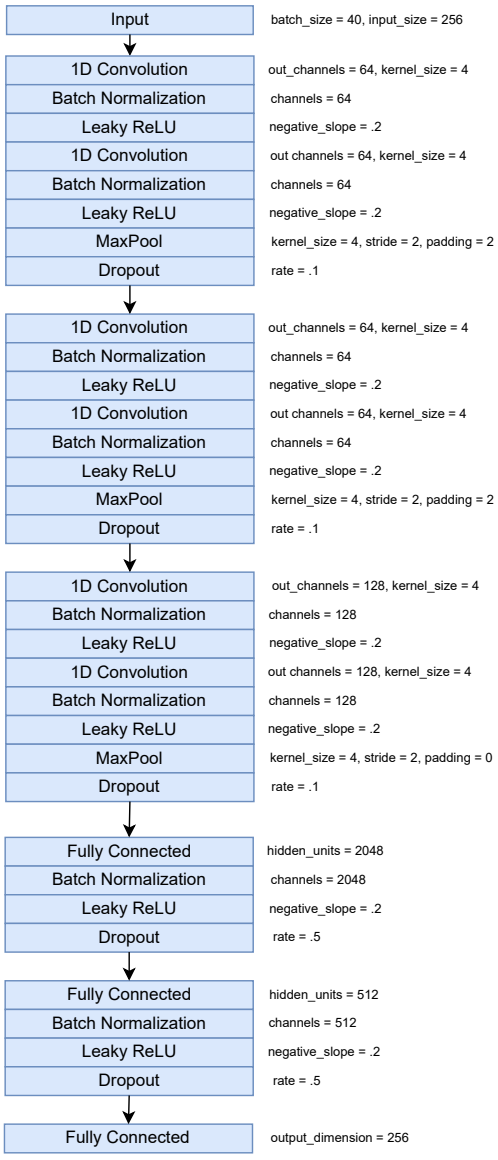

Figure 7.4: CNN architecture for discriminator in WF setting

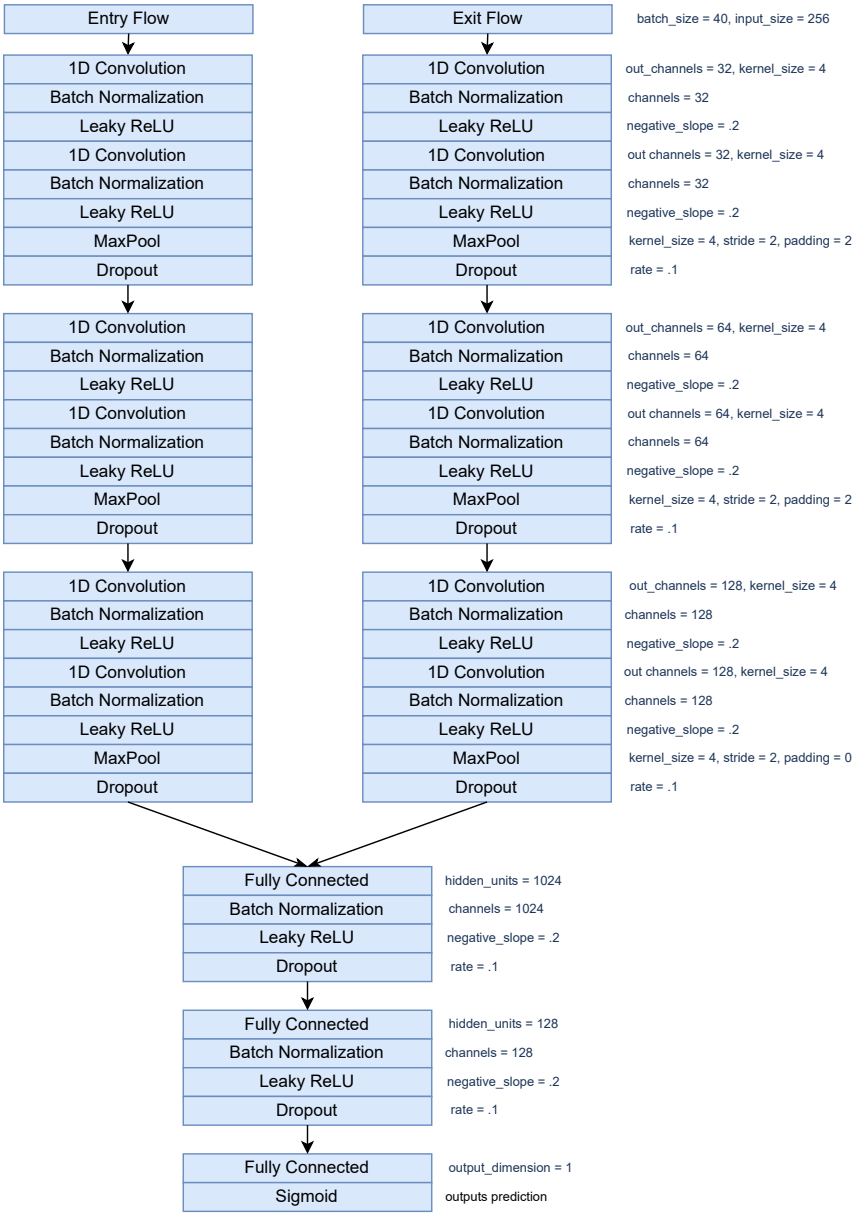

Figure 7.5: CNN architecture for discriminator in FC setting. Note that the entry and exit flows are processed separately before cominding for a final prediction.

7.4.2 Training and Model Distribution

In the WF setting, we train the embedder, discriminator, and generator using the Adam optimizer with a learning rate of .0001. The embedder uses PyTorch's learning rate scheduler ReduceLROnPlateau with minimum a learning rate of .000001. The dimensionality of the trace embeddings is 256 and the batch size is 40. The embedder's triplet loss uses Euclidean distance with a margin of 1. We train the discriminator-generator pair for 90 epochs.

The FC setting uses the same batch size and optimizer settings but trains the discriminator-generator pair for 60 epochs. We trained the models on an Nvidia GeForce RTX 3090 and find that it takes about an hour to train WF-DeTorrent and FC-DeTorrent. However, simulating DeTorrent on the chosen dataset and evaluating defense performance generally takes 3 to 4 more hours, making hyperparameter optimization prohibitively time-consuming. While using a dataset of 19,000 traces, the training process uses a maximum of 3 GiB of GPU memory.

With the trained models, we are able to use the GPU to generate a defended trace in about 140ms, making it possible to generate a defended dataset relatively quickly. The saved defense generator model is quite small at only 358 KiB as it has only about 86,400 trainable parameters.

One implementation challenge for DeTorrent is distributing the models. Because the upload padding is quite simple, only the generator needs to be distributed to bridges or relays. Our proposal for model distribution is similar to the one presented for the Surakav defense [36], which is that the semi-trusted Tor directory servers train, distribute, and occasionally update the defense generators. Given the small size of the DeTorrent generator, we don't expect the additional bandwidth to excessively burden the directory servers.

7.4.3 Real-World Implementation

To demonstrate the practicality of the DeTorrent defense, we implement it as a pluggable transport [95]. Pluggable transports (PTs) transform traffic between the client and a Tor bridge [118], which is a Tor relay that is not publicly listed. PTs are frequently used for censorship circumvention due to their ability to obfuscate traffic characteristics. We implemented DeTorrent on top of the WFPadTools framework [96], which extends the Obfsproxy PT [97] while adding functionality for common anti-traffic analysis strategies. The DeTorrent PT uses a private Tor bridge, which we run using Amazon Lightsail. Even though DeTorrent requires the LSTM generator to be repeatedly evaluated, we are able to defend five connections simultaneously with

Defenses	Bandwidth OH	Parameters
WTF-PAD	86.1%	normal_rcv, scaled 2x
FRONT	90.2%	$N_s = N_c = 1700, W_{min} = 1, W_{max} = 14$
BAP	90.2%	$\alpha = 2000, \sigma = 0, \mu = 0$
WF-DeTorrent	88.8%	$N_{download} = 1500$

Table 7.1: Defense Overheads and Parameters on the DF dataset

Defenses	Bandwidth OH	Parameters
WTF-PAD	86.8%	normal_rcv, scaled 6x
BAP	94.4%	$\alpha = 5000, \sigma = 0, \mu = 0$
Spring	102.7%	see [32]
Interspace	96.7%	see [32]
FRONT	95.0%	$N_s = N_c = 4000, W_{min} = 1, W_{max} = 14$
WF-DeTorrent	98.9%	$N_{download} = 3000$

Table 7.2: Defense Overheads and Parameters on the BE dataset

a modest virtual private server with 4GB RAM and 2 virtual CPUs running at less than 50% capacity. Thus, DeTorrent is not too computationally intensive for real-time use and does not require a GPU for implementation.

7.5 Website Fingerprinting Results

7.5.1 Simulated Defense Results

Defense performance on the BE dataset is shown in Table 7.4. On the BE dataset, we find that WF-DeTorrent reduces both the Tik-Tok and DF attacks to the lowest accuracy at 31.9% and 30.0% respectively. Because WF-DeTorrent is trained to 'trick' the discriminator into outputting inaccurate embeddings of the target traffic by making traffic from different web pages indistinguishable, we believe it is able to effectively mask the differences between many of the subpages. Thus, the WF attacks are usually unable to uniquely identify a given subpage in the BE dataset.

FRONT is the next-best performing defense with Tik-Tok and DF accuracies of 42.4% and

Defenses	Tik-Tok	DF
Undefended	97.7%	98.1%
BAP	94.4%	86.8%
WTF-PAD	92.1%	90.3%
FRONT	85.2%	79.7%
WF-DeTorrent	79.5%	74.5%

Table 7.3: Closed-World Accuracy on DF

Defenses	Tik-Tok	DF
Undefended	93.4%	94.3%
BAP	92.4%	94.3%
WTF-PAD	62.4%	59.7%
Spring	51.4%	46.8%
Interspace	44.9%	38.1%
FRONT	42.4%	41.2%
WF-DeTorrent	31.9%	30.0%

Table 7.4: Closed-World Accuracy on BE

Defenses	Bandwidth OH	Parameters
WTF-PAD	98.6%	normal_rcv, scaled 3.2x
BAP	96.6%	$\alpha = 4000, \sigma = 0, \mu = 0$
FRONT	97.9%	$N_s = N_c = 3400, W_{min} = 1, W_{max} = 14$
Decaf	79.6%	$\omega = 5, \frac{v}{k} = 1$
FC-DeTorrent	97.3%	$N_{download} = 3200$

Table 7.5: Defense Overheads on DCF

41.2% respectively. FRONT likely performs well because the inherent randomness in the shape and volume of the dummy packet distribution makes training the WF attacks more difficult. This effect appears exacerbated by the fact that the BE dataset is relatively small and distributes the instances for each website across ten subpages.

WTF-PAD provides a moderate amount of defense by masking the large inter-packet delays, which allows for the traces to be more easily identified. However, WTF-PAD defends in a fairly consistent manner without much obfuscation of the volume of quick bursts of traffic. Thus, many of the traces can still be correctly classified.

The Spring defense outperforms WTF-PAD, showing that an 'evolved' circuit padding defense can be somewhat effective. However, the Spring defense strategy is relatively consistent across traces, allowing for a DNN to effectively learn how to classify the defended traces. The Interspace defense, on the other hand, is 'probabilistically defined,' in the sense that it is initialized with one of a set of possible padding strategies at the client and relay. Interspace also

randomizes the parameters for the length and inter-arrival time distributions. As a result, Interspace is significantly more effective than its Spring counterpart. BAP, however, provides little defense against the Tik-Tok and DF attacks. This is likely because the BAP perturbation generator is trained to reduce the accuracy of a *static* attacker that cannot retrain. In this setting, the WF attacks can be retrained so that they are mostly unaffected by the BAP defense.

Defense results on the DF dataset are shown in Table 7.3. In terms of performance, the order of defenses is the same except that we are unable to evaluate Spring and Interspace, which require Tor logs to simulate. In general, the defenses did not reduce the accuracy nearly as much. However, this is likely due to the inherent difficulty of defending the DF dataset, which has 1,000 instances for each class, allowing for the deep learning-based WF attacks to better generalize. The DF dataset also uses only the home pages of the websites, meaning there is less intra-class variety. This especially reduces the effectiveness of defenses that utilize randomized dummy packet distributions, such as DeTorrent and FRONT. We believe that this is because this makes it much less likely that the defended trace for one class will be 'confused' with the trace of another class.

Using the recent Convolutional vision Transformer attack, discussed in Chapter 3, we find that using 10x data augmentation and the six different feature representations improves attack performance against a DeTorrent-defended BE dataset to 75.2%. However, this is still substantially lower than CvT's performance in the same configuration against the FRONT and Interspace defenses (see Table 3.3).

7.5.2 Trade-off Between Overhead and Performance

In order to fully evaluate the relationship between DeTorrent's performance and bandwidth overhead, we tested its ability to defend traffic in the BE dataset against the Tik-Tok attack for a variety of download volume ($N_{download}$) parameter choices. We varied $N_{download}$ from 1,000 to 7,000 while keeping the upload traffic ratio constant, resulting in overhead that varied from about 40% to about 210%, as shown in Figure 7.6. Tik-Tok attack accuracy was 52.8% for the lowest bandwidth setting and was reduced to 20.8%. We also find that increasing DeTorrent's bandwidth budget initially improves its performance, but is later subject to decreasing returns. To be specific, increasing $N_{download}$ from 1000 to 3000 decreases Tik-Tok performance by about 19.1%, while increasing $N_{download}$ from 5000 to 7000 only further decreases it by 4.9%. This is

likely due to the difficulty of defending particularly unique and high-volume traces.

Additionally, for comparison to the next-best non-scaled WF defense, we simulate FRONT on the BE dataset while using the original, non-scaled parameters. We find that this version of FRONT incurs a 59.5% bandwidth overhead while reducing Tik-Tok attack accuracy to 54.3%. For comparison, WF-DeTorrent with parameter $N_{download} = 1500$ defends BE with an overhead of 54.3% while reducing Tik-Tok attack accuracy to just 43.8%.

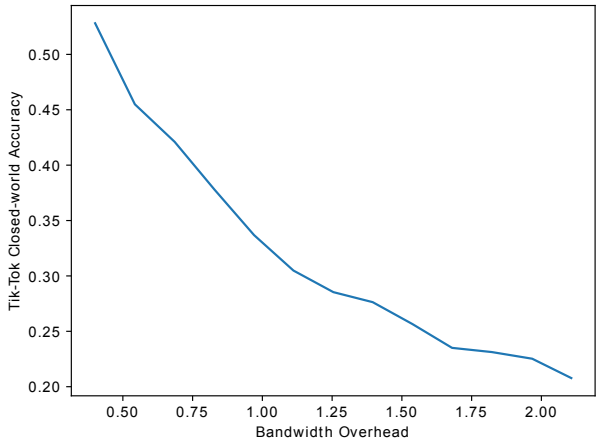

Figure 7.6: DeTorrent overhead vs. performance

7.5.3 Trace Representation Tuning

Because the choice of trace representation can significantly impact performance, we evaluate WF-DeTorrent's performance for different representation lengths and binning strategies. As shown in Figure 7.7, we find that the performance increases for larger trace representations until about length 512, likely because the granularity of the representation allows for WF-DeTorrent to make more fine-tuned strategies. However, WF-DeTorrent performance begins to decrease for larger representation sizes. We believe that this is due to the difficulty of accurately embedding the traces when bins are very small, as slight timing differences begin to change traffic representation significantly. As a result, we choose to use a trace representation of length 256, as it is associated with high performance while being less computationally intensive to train and implement compared to larger representations.

We also test a representation that spaces the bins evenly on a linear scale rather than on a logarithmic scale and find that this significantly decreases defense performance, reducing Tik-Tok to 43.2% rather than 31.9%. This decrease is likely because the linear scale does not as effectively differentiate between the frequent bursts of packets often seen early in a trace.

7.5.4 Defense Countermeasures

Since adversarial training allows the discriminator to adapt to the generator's strategy, we don't expect straightforward improvements to attacks based only on knowledge of the use of DeTorrent. Still, an attacker aware of the DeTorrent defense in use may use hyperparameter tuning to increase attack accuracy. During our WF defense evaluation, we modified the epoch number, input dimension, early stopping parameters, learning rate, and optimizer type. We find that the attack accuracy is highest when setting the number of epochs to 100, increasing the input dimension to 10,000, and setting the early stopping patience value to 10. Similarly, we modify the number of epochs used in the DCF attack to 1,500 and check validation accuracy to best improve performance. However, changing the other hyperparameters in the WF and FC settings did not significantly change performance.

However, one countermeasure that an attacker can use to better evade a defense is to generate multiple defended traces for every real trace in the dataset, as demonstrated in an evaluation of a variety of WF defenses by Matthews et al. [68]. This works particularly well when using small datasets and against defenses that use randomization, likely because it aids attacker model generalization and provides a better view of the variety of ways in which traffic may be defended. To test this, we generate a '10x' dataset where each trace in the training set is defended 10 times each (with re-sampled noise) to create a much larger training set. We find that this improves Tik-Tok attack performance from 31.9% to 48.2%. Thus, a motivated attacker would likely use this countermeasure to best attack DeTorrent (as well as comparable defenses such as FRONT).

7.5.5 Real-World Implementation Results

Then, we crawl a set of popular websites through the pluggable transport to test current website fingerprinting defenses against the collected packet traces. To test the transferability of the defense, the DeTorrent download padding distribution is determined using a generator trained

using the BE dataset and scaled to achieve a similar bandwidth overhead. As a result, the generator is not fine-tuned to current internet traffic. However, since DeTorrent's defense performance transfers well to different settings and sets of websites, we find that defense performance remains high, reducing Tik-Tok to a 21.5% accuracy with an 88.8% bandwidth overhead.

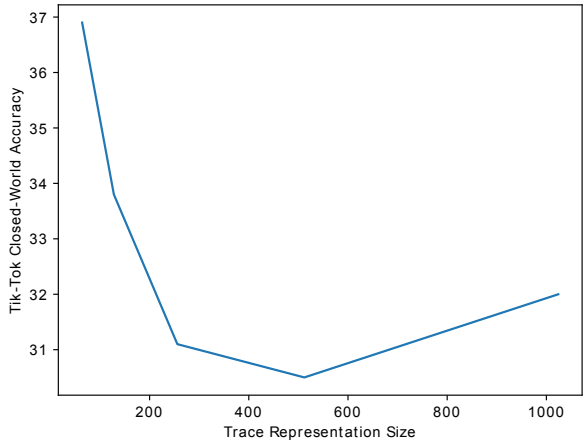

Figure 7.7: DeTorrent performance vs. representation Length

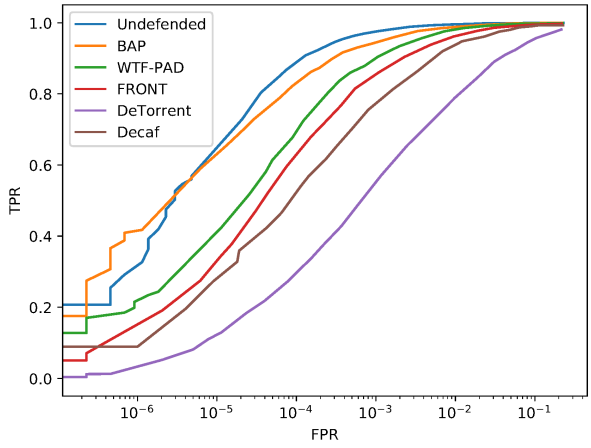

Figure 7.8: Flow correlation defense performance against DeepCoFFEA. Note that lower TPR implies a better defense.

7.6 Flow Correlation Results

To ensure a fair comparison between the chosen defenses, we scale them to operate with comparable bandwidth overhead, as shown in Table 7.5. Also, one of the main challenges of the flow correlation setting is achieving a high enough true positive rate without allowing the number of false positives to outnumber the true positives. As a result, it is most sensible to evaluate flow correlation defenses by comparing their true positive and false positive rates at various DeepCoFFEA correlation thresholds; this is illustrated in Figure 7.8.

In terms of reducing DeepCoFFEA's performance, FC-DeTorrent far outperforms the compared defenses, as shown in Figure 7.8. While other defenses allow a high true positive rate for associated false positive rates of as little as 10^{-3}, FC-DeTorrent noticeably reduces DeepCoFFEA's true positive rate for all but the highest false positive rates. For a 10^{-4} false positive rate, FC-DeTorrent reduced DeepCoFFEA's true positive rate to .30, compared to about .54 for Decaf and .63 for FRONT. For a 10^{-5} false positive rate, the true positive rates are about .13 for FC-DeTorrent, .30 for Decaf, .34 for FRONT, and .42 for WTF-PAD. FC-DeTorrent performance is likely due to its ability to learn how to pad entry flows such that they are easily confused with non-associated entry flows, thus confusing the discriminator. Also, FC-DeTorrent's defense advantage over the alternate FC defenses is significantly larger than WF-DeTorrent's advantage in the WF setting: we believe that this is because FC-DeTorrent can directly train against the discriminator, while WF-DeTorrent must rely on the embedder to estimate descriminator performance.

FRONT provides a reasonable degree of FC defense, likely because it's randomized and because it sends traffic according to the Rayleigh distribution, which mimics the distribution of real traffic. As a result, the defended entry traffic associated with a given exit flow is likely to look similar to other defended entry flows. The same can be said of Decaf, which defends the real traffic using traffic patterns found in previously collected traffic. However, FRONT's use of the Rayleigh distribution makes the padding somewhat predictable, as the amount of padding will always increase until it peaks and will then slowly taper off. As a result, DeepCoFFEA is able to learn how to effectively ignore much of the padding. Decaf, on the other hand, avoids this problem by using more realistic traffic patterns and therefore is a more performant defense.

WTF-PAD obfuscated some of the inter-packet delays that DeepCoFFEA uses to identify traffic; as a result, it provides some defense as well. Still, WTF-PAD does not place much

emphasis on the more coarse features of the traffic, such as the sizes of the largest traffic bursts. As a result, DeepCoFFEA is still able to correlate many of the flows defended by WTF-PAD. Because BAP was trained to provide adversarial perturbations against a static model, DeepCoFFEA's feature embedders can retrain against the BAP-defended dataset. As a result, BAP's performance is lower than that of other defenses.

7.7 Discussion

7.7.1 Transferability

Across Different Datasets

It is important to ensure that DeTorrent is able to effectively defend traffic with different characteristics than traffic in the training datasets. Otherwise, DeTorrent's performance may degrade rapidly over time or while defending new websites. Accordingly, we test DeTorrent's transferability by conducting an experiment where we train WF-DeTorrent on traces associated with one set of websites while testing the defense on traces associated with a *different* set.

More specifically, we partition the BE dataset so that the first partition contains 20 instances of 10 subpages of 50 of the websites, while the second partition contains the same number of instances of the other 45 websites. We refer to the first partition as BE-1 and the second partition as BE-2. Then, we train the WF-DeTorrent defense using only the BE-1 dataset and simulate the defense on BE-2 (which does not contain any traces *or* websites that WF-DeTorrent was trained on) with $N_{download} = 3000$. We find that the defense drops Tik-Tok's accuracy to 40.3%, showing that it was effective. Also, note that the higher attack accuracy is likely because the attacker is only choosing from 50 websites, rather than 95.

To set a baseline for how well WF-DeTorrent performs when training and testing on the same dataset, we both train and test it on BE-2 with the same amount of overhead. We find that it decreases Tik-Tok's performance to 39.6%. Because this accuracy is only .7% lower than the accuracy found when WF-DeTorrent was trained on BE-1, we conclude that the WF-DeTorrent framework generalizes well and does not simply memorize fine-grained patterns in the training data. Accordingly, we expect the defense provided by WF-DeTorrent to be highly transferable.

Against Different Attacks

In this paper, we describe two different training frameworks with one defending against website fingerprinting and the other defending against flow correlation. However, one might like to defend against both types of attacks. To test how DeTorrent's performance against one attack transfers to the other, we test how well WF-DeTorrent functions in the FC setting and vice versa. To train the WF and FC DeTorrent models, we use the BE and DCF datasets respectively. Note that WF-DeTorrent and FC-DeTorrent are trained on entirely different datasets collected using different methodologies and at different points in time. As a result, the models used in these experiments are unlikely to be fine-tuned in terms of defense performance, and the above results should be taken as a lower bound of what performance is possible.

We find that FC-DeTorrent provides some defense in the WF setting, though its performance is degraded. To be specific, it reduces Tik-Tok performance on the BE dataset to 45.8%. This represents a 12.9% increase in Tik-Tok accuracy, putting it behind Decaf, FRONT, and Interspace in terms of performance, though it still outperforms BAP, WTF-PAD, and Spring.

On the other hand, WF-DeTorrent performs only slightly worse than the FC setting, with a TPR of about .32 for a FPR of 10^{-4} and a TPR of about .58 for a 10^{-3} FPR, compared to FC-DeTorrent's TPRs of .30 and .57 respectively. Therefore, we find that WF-DeTorrent is transferable in the sense that it functions as an effective defense in both the WF and FC settings; accordingly, it may be a reasonable choice for users concerned about both types of attacks.

7.7.2 Overhead

While DeTorrent doesn't add any latency of its own, past work by Witwer et al. on network-wide Tor simulation [100] has demonstrated that even zero-delay defenses likely strain the Tor network and incur latency. Their study used a 'time-to-nth-byte' metric, finding that defenses Spring and Interspace had median latency overheads of 1.6% and 7.8% for 1MiB downloads compared to undefended downloads. Since the Spring and Interspace defenses avoid delaying traffic and incur similar bandwidth overhead compared to DeTorrent, as shown in Tables 7.1 and 7.2, we might expect that DeTorrent would have similar results, though low observed bandwidth overhead and high failure rates in the experiments make precise estimation difficult. Still, it's quite possible that widespread adoption of DeTorrent would lead to non-negligible latency overhead.

7.7.3 Implementation Framework

While the PT system is convenient due to its relative flexibility in terms of supporting various traffic analysis defenses [95], it only obfuscates the traffic between the client and the bridge. While this protects against local eavesdroppers, it leaves the user vulnerable to malicious bridges.

Tor has implemented a circuit padding framework [33], which adds dummy cells into circuit traffic to protect against WF attacks. Because the padding can be sent between the client and any relay in the circuit, it can be configured to stop at a middle node and thus protect against a malicious guard. However, the circuit padding framework is limited as it cannot delay traffic and must add dummy traffic based on pre-defined state machines. As a result, it is inflexible in terms of defense implementation and does not natively support randomized padding or complex pattern recognition, preventing it from being able to support DeTorrent.

7.8 Summary

This chapter introduces DeTorrent, an adversarial padding-only traffic analysis defense designed to protect Tor traffic from WF and FC attacks. Traditional defenses often require additional infrastructure or induce significant latency, but DeTorrent utilizes dummy traffic to obfuscate data without explicit delays, adhering to Tor developers' recommendations against excessive latency.

DeTorrent leverages adversarial techniques, inspired by generative adversarial networks, where a generator neural network creates padding strategies to confuse an attacker network. This setup ensures the defense remains robust against retraining by attackers. The generator is implemented as an LSTM network, enabling real-time decision-making for padding.

The chapter details DeTorrent's architecture, including the binned representation of traffic traces to facilitate tensor operations and differentiable updates. The WF setting employs an embedder to map traces to a low-dimensional space, optimizing the generator's strategy through triplet loss. For FC, the generator directly reduces the discriminator's correlation accuracy.

Experiments show that DeTorrent significantly reduces attack success rates compared to other defenses, achieving substantial reductions in accuracy for leading WF attacks like Tik-Tok and Deep Fingerprinting. The defense maintains high performance with reasonable bandwidth overhead and shows robustness in real-world implementations as a Tor pluggable transport.

Overall, DeTorrent demonstrates that effective traffic analysis defense can be achieved solely with dummy traffic, providing a practical solution for enhancing user privacy on Tor without compromising network performance.

Chapter 8

Conclusion and Discussion

This dissertation has explored the significant threat posed by website fingerprinting (WF) and flow correlation (FC) attacks, demonstrating how these attacks compromise the privacy and anonymity that Tor aims to provide. By improving upon previous work and presenting novel approaches for both attacks and defenses, this work contributes to the ongoing effort to safeguard online anonymity.

Firstly, we have improved upon existing WF and FC attacks to ensure high performance even in challenging and realistic settings. By incorporating improved data augmentation, feature representation, and deep learning model architectures, we have shown that these attacks can achieve higher accuracy and reliability, particularly in large open-world scenarios. Our precision optimization techniques further enhance the attacker's ability to maintain performance in realistic and challenging settings.

Secondly, we validated our techniques using a realistic dataset of genuine Tor traffic, confirming their applicability and effectiveness in real-world environments. This step addresses a common critique in WF research regarding the use of synthetic datasets and uncertain base rates, providing a more robust evaluation of our methods.

In response to the improved attack capabilities, we developed two novel defenses: RegulaTor and DeTorrent. RegulaTor leverages common patterns in web browsing traffic to regularize traffic traces, significantly reducing the effectiveness of WF attacks. However, recognizing that RegulaTor does not defend against FC attacks and introduces delays, we designed DeTorrent to mitigate both WF and FC attacks without impacting traffic latency. DeTorrent utilizes a long

short-term memory network to generate dummy traffic insertion strategies that resist adversarial retraining, providing a comprehensive defense mechanism for Tor users.

Through these contributions, this dissertation supports the thesis that while website fingerprinting and flow correlation attacks pose a real threat to Tor users, the risk can be mitigated by implementing effective traffic analysis defenses. Our findings highlight the importance of continuous innovation in attack and defense strategies.

8.1 Future Work

8.1.1 Attacks on Genuine Tor Traffic

While this dissertation successfully demonstrated WF attacks and precision optimization techniques against real-world traffic, further investigation is warranted to better understand the extent of this vulnerability. For example, it is not immediately clear why some websites are very difficult to identify, while others can be recognized with high precision and recall. Potential reasons include that the difficult-to-identify websites may be very simple, very similar to other websites, or have many popular subpages that vary significantly from one another. Either way, a better understanding of why certain websites are more susceptible may aid in the development of traffic analysis defenses. Overall, while previous work on WF attacks on genuine Tor traffic has had limited success [40], we suspect that the combination of updated WF attack approaches, precision optimization techniques, and the high volume of data available in the GTT23 dataset [9] helps us to achieve improved attack performance.

Another related and avenue of investigation for real-world WF attacks is how an attacker could combine genuine Tor traffic with synthetic datasets that focus on the less commonly visited websites of interest. This way, the attacker would be able to understand the distribution of traffic sent through Tor and the base rate of targeted websites while also fixing the problem of having limited data for less-visited websites. Testing the effectiveness of WF attacks against onion service sessions in the real world is also an interesting avenue for future research.

8.1.2 Traffic Analysis Defense Implementation

In recent years, several effective traffic analysis defenses have been proposed [36, 38, 31, 43], in addition to the techniques mentioned in this dissertation. However, implementing these defenses

for the mass of Tor users, which may include modifications to the Tor network or continued maintenance in the form of tuning the defenses, remains a significant challenge. While Tor's circuit padding framework [33] can support defenses that operate in the style of WTF-PAD [30], resisting more recent traffic analysis attacks will likely require either splitting or delaying traffic. In fact, few padding-only defenses provide significant protection against the CvT attack presented in this dissertation.

Pluggable transport implementations [74, 96] are useful for prototyping defenses and potentially protecting especially careful Tor users. However, they may not be easily scalable in terms of defending a large number of users. Another promising approach is the Maybenot framework [44], which extends and generalizes Tor's Circuit Padding Framework. Defenses can be implemented in this framework as probabilistic state machines that can both add padding and block traffic from being sent. The Maybenot framework also includes a defense simulator to aid in the development of future traffic analysis defenses.

8.1.3 Impact of Overhead on User Experience

Existing traffic analysis defenses typically induce overhead in the form of extra bandwidth or increased latency. While these can be measured through simulations or prototype implementations, such as with pluggable transports, it is more challenging to predict how a widespread roll-out of these defenses would impact the Tor network. For example, recent work uses Shadow [119] to simulate a version of Tor employing padding-only WF defenses. The study finds that the increased queue lengths caused by these defenses leads to delays, despite the defenses not being designed to delay traffic. Therefore, it may be beneficial for future evaluations of traffic analysis defenses to include network-wide simulations of Tor to better understand potential impacts on user experience. Given the recent availability of datasets of genuine Tor traffic, it may also be beneficial for defenses to use them to estimate their bandwidth and latency overheads, rather than relying on estimates based on synthetic datasets.

Furthermore, predicting how a specific amount of latency (e.g., 10% longer web page load times) affects user experience is challenging, partly due to the lack of a well-defined user model. Future user studies could measure how increased latency impacts Tor's usability and identify the trade-offs users are willing to make between security and ease of use. Alternatively, future research could investigate methods to improve Tor's web page loading times, potentially building

off efforts by Wang et al. to reduce the number of round trips required for loading each page [120].

References

[1] Mike Perry. A critique of website traffic fingerprinting attacks. `https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks`, 2013. Accessed: 2020-11-17.

[2] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 753–770, Boston, MA, August 2022. USENIX Association.

[3] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 263–274, 2014.

[4] Tao Wang and Ian Goldberg. On Realistically Attacking Tor with Website Fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2016(4):21–36, 2016.

[5] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1131–1148, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Meng Shen, Kexin Ji, Zhenbo Gao, Qi Li, Liehuang Zhu, and Ke Xu. Subverting website fingerprinting defenses with robust traffic representation. In *32th USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 2023.

[7] Se Oh, Nate Mathews, Mohammad Saidur Rahman, Matthew Wright, and Nicholas Hopper. Gandalf: Gan for data-limited fingerprinting. volume 2021, 12 2020.

[8] Mohammad Saidur Rahman, Payap Sirinam, Nate Matthews, Kantha Girish Gangadhara, and Matthew Wright. Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks. *Proceedings of Privacy Enhancing Technologies*, 2020.

[9] Rob Jansen, Ryan Wails, and Aaron Johnson. A measurement of genuine tor traces for realistic website fingerprinting, 2024, 2404.07892.

[10] Se Eun Oh, Taiji Yang, Nate Mathews, James K Holland, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. Deepcoffea: Improved flow correlation attacks on tor via metric learning and amplification. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1915–1932. IEEE, 2022.

[11] Tor Project. Tor metrics. `https://metrics.torproject.org/userstats-relay-country.html`, 2022. Accessed: 2022-09-20.

[12] Andrew Hintz. Fingerprinting websites using traffic analysis. In *Proceedings of the 2nd International Conference on Privacy Enhancing Technologies*, PET'02, page 171–178, Berlin, Heidelberg, 2002. Springer-Verlag.

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

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

[15] Tao Wang and Ian Goldberg. Improved website fingerprinting on Tor. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 201–212, 2013.

[16] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. *23rd USENIX Security Symposium (USENIX Security 14)*, pages 143–157, 2014.

[17] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, Austin, TX, August 2016. USENIX Association.

[18] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. Website Fingerprinting at Internet Scale. (February):21–24, 2017.

[19] Payap Sirinam, Marc Juarez, Mohsen Imani, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1928–1943, 2018.

[20] Se Eun Oh, Saikrishna Sunkam, and Nicholas Hopper. p1-FP: Extraction, Classification, and Prediction of Website Fingerprints with Deep Learning. *Proceedings on Privacy Enhancing Technologies*, 2019(3):191–209, 2019, arXiv:1711.03656v2.

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

[22] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated Website Fingerprinting through Deep Learning. 2018.

[23] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proceedings on Privacy Enhancing Technologies*, 2019(4):292–310, oct 2019.

[24] Tobias Pulls and Rasmus Dahlberg. Website fingerprinting with website oracles. *Proceedings on Privacy Enhancing Technologies*, 2020:235–255, 01 2020.

[25] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE Symposium on Security and Privacy*, pages 332–346, 2012.

[26] Xiang Cai, Rishab Nithyanand, and Rob Johnson. Cs-buflo: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, page 121–130, New York, NY, USA, 2014. Association for Computing Machinery.

[27] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 227–238. Association for Computing Machinery, nov 2014.

[28] Charles V Wright, Scott E Coull, and Fabian Monrose. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. Technical report, 2009.

[29] Rishab Nithyanand, Xiang Cai, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 131–134. Association for Computing Machinery, nov 2014.

[30] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. WTF-PAD: Toward an Efficient Website Fingerprinting Defense for Tor. *Computing Research Repository (CoRR)*, abs/1512.0, 2015, 1512.00524.

[31] Jiajun Gong and Tao Wang. Zero-delay lightweight defenses against website fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 717–734. USENIX Association, August 2020.

[32] Tobias Pulls. Towards effective and efficient padding machines for tor. 2022.

[33] Mike Perry and George Kadianakis. Circuit padding developer documentation. `https://github.com/torproject/tor/blob/master/doc/HACKING/CircuitPaddingDevelopment.md`, 2021. Accessed: 2022-09-17.

[34] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew K. Wright. Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces. *IEEE Transactions on Information Forensics and Security*, 16:1594–1609, 2021.

[35] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Defeating DNN-Based traffic analysis systems in Real-Time with blind adversarial perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2705–2722. USENIX Association, August 2021.

[36] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. Surakav: Generating realistic traces for a strong website fingerprinting defense. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1558–1573. IEEE, 2022.

[37] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017.

[38] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. Trafficsliver: Fighting website fingerprinting attacks with traffic splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1971–1985, New York, NY, USA, 2020. Association for Computing Machinery.

[39] Sébastien Henri, Gines Garcia-Aviles, P. Serrano, A. Banchs, and P. Thiran. Protecting against website fingerprinting with multihoming. *Proceedings on Privacy Enhancing Technologies*, 2020:89 – 110, 2020.

[40] Giovanni Cherubin. Bayes, not Naïve: Security Bounds on Website Fingerprinting Defenses. *Proceedings on Privacy Enhancing Technologies*, 2017(4):215–231, oct 2017, 1702.07707.

[41] Shuai Li, Huajun Guo, and Nicholas Hopper. Measuring information leakage in website fingerprinting attacks and defenses. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1977–1992, 2018.

[42] Alex Veicht, Cedric Renggli, and Diogo Barradas. Deepse-wf: Unified security estimation for website fingerprinting defenses. *Proceedings on Privacy Enhancing Technologies*, 2023(2), 2023.

[43] Jean-Pierre Smith, Luca Dolfi, Prateek Mittal, and Adrian Perrig. QCSD: A QUIC Client-Side Website-Fingerprinting defence framework. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 771–789, Boston, MA, August 2022. USENIX Association.

[44] Tobias Pulls and Ethan Witwer. Maybenot: A framework for traffic analysis defenses. In *Proceedings of the 22nd Workshop on Privacy in the Electronic Society*, WPES '23, page 75–89, New York, NY, USA, 2023. Association for Computing Machinery.

[45] Jean-François Raymond. *Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems*, pages 10–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[46] Xinyuan Wang, Douglas S. Reeves, and S. Felix Wu. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In Dieter Gollmann, Günther Karjoth, and Michael Waidner, editors, *Computer Security — ESORICS 2002*, pages 244–263, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[47] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew Wright. Timing attacks in low-latency mix systems. In Ari Juels, editor, *Financial Cryptography*, pages 251–265, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[48] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation onion router. In *13th USENIX Security Symposium (USENIX Security 04)*, San Diego, CA, August 2004. USENIX Association.

[49] S.J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 183–195, 2005.

[50] L. Overlier and P. Syverson. Locating hidden servers. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15 pp.–114, 2006.

[51] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *Computer Security – ESORICS 2006*, pages 18–33, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[52] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. On flow correlation attacks and countermeasures in mix networks. In David Martin and Andrei Serjantov, editors, *Privacy Enhancing Technologies*, pages 207–225, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[53] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. RAPTOR: Routing attacks on privacy in tor. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 271–286, Washington, D.C., August 2015. USENIX Association.

[54] Matthew Edman and Paul Syverson. As-awareness in tor path selection. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 380–389, New York, NY, USA, 2009. Association for Computing Machinery.

[55] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. Lastor: A low-latency as-aware tor client. *IEEE/ACM Trans. Netw.*, 22(6):1742–1755, dec 2014.

[56] Joshua Juen, Anupam Das, Aaron Johnson, Nikita Borisov, and Matthew Caesar. Defending tor from network adversaries: A case study of network path prediction. *CoRR*, abs/1410.1823, 2014, 1410.1823.

[57] Y. Sun, A. Edmundson, N. Feamster, M. Chiang, and P. Mittal. Counter-raptor: Safeguarding tor against active routing attacks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 977–992, Los Alamitos, CA, USA, may 2017. IEEE Computer Society.

[58] Armon Barton and Matthew Wright. Denasa: Destination-naive as-awareness in anonymous communications. *Proceedings on Privacy Enhancing Technologies*, 2016, 02 2016.

[59] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *Proceedings of CCS 2007*, October 2007.

[60] arma. Research problem: better guard rotation parameters, 2011. Accessed: 2022-7-14.

[61] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. DeepCorr. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, jan 2018.

[62] Vera Rimmer, Theodor Schnitzler, Tom Van Goethem, Abel Romero, Wouter Joosen, and Katharina Kohls. Trace oddity: Methodologies for data-driven traffic analysis on tor. *Proceedings on Privacy Enhancing Technologies*, 2022:314–335, 07 2022.

[63] Daniela Lopes, Jin-Dong Dong, Daniel Castro, Pedro Medeiros, Diogo Barradas, Bernardo Portela, João Vinagre, Bernardo Ferreira, Nicolas Christin, and Nuno Santos. Flow correlation attacks on tor onion service sessions with sliding subset sum. *Proceedings 2024 Network and Distributed System Security Symposium*, 2024.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems*, 2017.

[65] Tobias Pulls. Towards effective and efficient padding machines for Tor. 2020, 2011.13471.

[66] Nate Matthews, James K. Holland, Nicholas Hopper, and Matthew Wright. Evolving website fingerprinting attacks with attention and multi-channel feature representation, 2024.

[67] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y. Zhao. Patch-based defenses against web fingerprinting attacks. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, AISec '21, page 97–109, New York, NY, USA, 2021. Association for Computing Machinery.

[68] Nate Matthews, James Holland, Se Eun Oh, Mohammad Saidur Rahman, Matthew Wright, and Nicholas Hopper. Sok: A critical evaluation of efficient website fingerprinting defenses. In *2023 IEEE Symposium on Security and Privacy*. IEEE, 2023.

[69] Sandra Siby, Ludovic Barman, Christopher Wood, Marwan Fayed, Nick Sullivan, and Carmela Troncoso. Evaluating practical quic website fingerprinting defenses for the masses. *Proceedings on Privacy Enhancing Technologies*, 2023:79–95, 10 2023.

[70] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019, 1711.05101.

[71] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[72] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. In *arXiv preprint arXiv:1706.02677*, 2017.

[73] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[74] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. Wfdefproxy: Real world implementation and evaluation of website fingerprinting defenses. *IEEE Transactions on Information Forensics and Security*, 19:1357–1371, 2024.

[75] N. Mathews, J. K. Holland, S. Oh, M. Rahman, N. Hopper, and M. Wright. Sok: A critical evaluation of efficient website fingerprinting defenses. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 969–986, Los Alamitos, CA, USA, may 2023. IEEE Computer Society.

[76] Tao Wang. High precision open-world website fingerprinting. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 152–167, 2020.

[77] Yanbin Wang, Haitao Xu, Zhenhao Guo, Zhan Qin, and Kui Ren. snwf: Website fingerprinting attack by ensembling the snapshot of deep learning. *IEEE Transactions on Information Forensics and Security*, 17:1214–1226, 2022.

[78] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings 2019 Network and Distributed System Security Symposium*, NDSS 2019. Internet Society, 2019.

[79] Yin Zhang and Vern Paxson. Detecting stepping stones. In *USENIX Security Symposium*, 2000.

[80] David L. Donoho, Ana Georgina Flesia, Umesh Shankar, Vern Paxson, Jason Coit, and Stuart Staniford-Chen. Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In *International Symposium on Recent Advances in Intrusion Detection*, 2002.

[81] Kunikazu Yoda and Hiroaki Etoh. Finding a connection chain for tracing intruders. In Frédéric Cuppens, Yves Deswarte, Dieter Gollmann, and Michael Waidner, editors, *Computer Security - ESORICS 2000*, pages 191–205, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[82] Xinyuan Wang, Douglas S. Reeves, and Shyhtsun Felix Wu. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *European Symposium on Research in Computer Security*, 2002.

[83] Ting He and Lang Tong. Detecting encrypted stepping-stone connections. *IEEE Transactions on Signal Processing*, 55(5):1612–1623, 2007.

[84] Avrim Blum, Dawn Song, and Shobha Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. In Erland Jonsson, Alfonso Valdes, and Magnus Almgren, editors, *Recent Advances in Intrusion Detection*, pages 258–277, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[85] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015.

[86] Lennart Oldenburg, Marc Juarez, Enrique Argones-Rúa, and Claudia Diaz. Mixmatch: Flow matching for mixnet traffic. *Proc. Priv. Enhancing Technol.*, 2024:276–294, 2024.

[87] James Holland and Nate Matthews. Ssid: Stepping stone intrusion detection. `https://github.com/jkhollandjr/SSID`, 2024. Accessed: 2024-06-11.

[88] Xiapu Luo, Peng Zhou, Edmond W W Chan, Wenke Lee, Rocky K C Chang, and Roberto Perdisci. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. Technical report, 2011.

[89] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: attacks and defenses. Technical report, 2006.

[90] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q.

Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[91] hyperopt. Hyperopt: Distributed hyperparameter optimization. `https://github.com/hyperopt/hyperopt`, 2022. Accessed: 2021-5-21.

[92] Payap Sirinam. Website fingerprinting using deep learning. *Thesis. Rochester Institute of Technology*, 2019.

[93] webfp. Tor browser crawler. `https://github.com/webfp/tor-browser-crawler`, 2017. Accessed: 2021-11-17.

[94] Defenses notes. `http://home.cse.ust.hk/~taow/wf/defenses/`. Accessed: 2020-9-21.

[95] Tor Project. Tor: Pluggable transports. `https://2019.www.torproject.org/docs/pluggable-transports.html.en`, 2022. Accessed: 2022-5-21.

[96] Marc Juarez. Wfpadtools. `https://github.com/mjuarezm/wfpadtools`, 2015. Accessed: 2021-5-21.

[97] Tor Project. Pluggable transport for obfuscated traffic. `https://gitweb.torproject.org/pluggable-transports/obfsproxy.git`, 2014. Accessed: 2022-7-21.

[98] Report: State of the web. `https://httparchive.org/reports/state-of-the-web#bytesTotal`. Accessed: 2021-9-21.

[99] Torspec: Padding negotiation. `https://gitweb.torproject.org/torspec.git/tree/proposals/254-padding-negotiation.txt`. Accessed: 2021-9-21.

[100] Ethan Witwer, James K. Holland, and Nicholas Hopper. Padding-only defenses add delay in tor. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society*, WPES'22, page 29–33, New York, NY, USA, 2022. Association for Computing Machinery.

[101] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2805–2824, 2019.

[102] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[103] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997, https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf.

[104] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2016.

[105] James K Holland and Nicholas Hopper. RegulaTor: A Straightforward Website Fingerprinting Defense. *Proceedings on Privacy Enhancing Technologies*, 2022, 2022.

[106] NumPy Reference. numpy.geomspace, 2023. Accessed: 2023-7-07.

[107] Hao Jiang and Constantinos Dovrolis. Why is the internet traffic bursty in short time scales? *SIGMETRICS Perform. Eval. Rev.*, 33(1):241–252, jun 2005.

[108] M.E. Crovella and A. Bestavros. Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[109] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2053–2069, New York, NY, USA, 2017. Association for Computing Machinery.

[110] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. Non-blind watermarking of network flows. *IEEE/ACM Transactions on Networking*, 22, 03 2012.

[111] Ilmari Karonen. Estimating rate of occurrence of an event with exponential smoothing and irregular events, 2014. Accessed: 2022-8-13.

[112] Open PageRank Initiative. What is open pagerank? `https://www.domcop.com/openpagerank/what-is-openpagerank`, 2022. Accessed: 2022-08-01.

[113] Amazon. Alexa top 1m. `http://s3.amazonaws.com/alexa-static/top-1m.csv.zip`, 2022. Accessed: 2022-08-01.

[114] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright. Wtf-pad, 2022. Accessed: 2022-7-14.

[115] Tobias Pulls. Tor circuit padding simulator, 2019. Accessed: 2022-7-14.

[116] Jiajun Gong and Tao Wang. *WebsiteFingerprinting*. `https://github.com/websitefingerprinting/WebsiteFingerprinting`, 2022. Accessed: 2022-7-14.

[117] SPIN-UMass. Blanket, 2022. Accessed: 2022-7-14.

[118] Tor Project. Get bridges for tor, 2022. Accessed: 2022-7-28.

[119] Rob Jansen and Nicholas Hopper. Shadow: Running tor in a box for accurate and efficient experimentation. In *Network and Distributed System Security Symposium*, 2011.

[120] Tao Wang. Designing a better browser for tor with blast. In *Proceedings 2020 Network and Distributed System Security Symposium*, NDSS 2020. Internet Society, 2020.