DeTorrent: An Adversarial Padding-only Traffic Analysis Defense

James K Holland
University of Minnesota
Minneapolis, USA
holla556@umn.edu

Jason Carpenter
University of Minnesota
Minneapolis, USA
carpe415@umn.edu

Se Eun Oh
Ewha Womans University
Seoul, South Korea
seoh@ewha.ac.kr

Nicholas Hopper
University of Minnesota
Minneapolis, USA
hoppernj@umn.edu

ABSTRACT

While anonymity networks like Tor aim to protect the privacy of their users, they are vulnerable to traffic analysis attacks such as Website Fingerprinting (WF) and Flow Correlation (FC). Recent implementations of WF and FC attacks, such as Tik-Tok and DeepCoFFEA, have shown that the attacks can be effectively carried out, threatening user privacy. Consequently, there is a need for effective traffic analysis defense.

There are a variety of existing defenses, but most are either ineffective, incur high latency and bandwidth overhead, or require additional infrastructure. As a result, we aim to design a traffic analysis defense that is efficient and highly resistant to both WF and FC attacks. We propose DeTorrent, which uses competing neural networks to generate and evaluate traffic analysis defenses that insert 'dummy' traffic into real traffic flows. DeTorrent operates with moderate overhead and without delaying traffic. In a closed-world WF setting, it reduces an attacker's accuracy by 61.5%, a reduction 10.5% better than the next-best padding-only defense. Against the state-of-the-art FC attacker, DeTorrent reduces the true positive rate for a 10^{-5} false positive rate to about .12, which is less than *half* that of the next-best defense. We also demonstrate DeTorrent's practicality by deploying it alongside the Tor network and find that it maintains its performance when applied to live traffic.

KEYWORDS

website fingerprinting, traffic analysis, deep learning

1 INTRODUCTION

As people's personal and professional lives increasingly depend on the Internet, the threat of network surveillance has become particularly salient. To combat this threat, many users utilize privacy-enhancing tools such as Tor to protect themselves. Accordingly, Tor has surged in popularity, recording millions of users each day [39, 55]. Tor operates by encrypting traffic and routing it through a series of volunteer-run relays, ensuring that no single entity can observe both entry and exit traffic [14]. The encryption strategy, known as onion routing, is characterized by messages encapsulated by multiple layers of encryption. Each successive Tor relay then strips a layer of encryption from the message and forwards it to the revealed destination [63].

While data sent through Tor may be secure, the frequency and timing of the associated network traffic may be used by traffic analysis techniques. These techniques include website fingerprinting [1, 7, 10, 22, 24, 47, 50, 51, 61, 65, 69, 70, 75–77] and end-to-end flow correlation [37, 42, 48, 62, 68, 73, 84], which have been thoroughly discussed in past literature. Researchers have demonstrated that both can be carried out effectively.

In a website fingerprinting (WF) attack, an adversary between the user and the first Tor relay records the timing and volume information of the Tor traffic and determines which website the user is visiting. Because this attack requires only a passive, local adversary, it is particularly threatening to user privacy. Alternatively, in the flow correlation (FC) attack setting, the adversary records traffic metadata as it both enters and exits the Tor network. Then, the adversary attempts to correlate associated entry and exit flows to determine both the origin and destination of the traffic. This attack is explicitly mentioned in early literature; however, security against end-to-end attacks is difficult for low-latency anonymity networks and an explicit non-goal of the original Tor design [14].

A variety of traffic analysis defenses have been proposed to prevent website fingerprinting and flow correlation [10, 11, 18, 20, 27, 33, 38, 43, 58, 66, 78, 82]. These defenses 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 [81], we avoid explicitly delaying packets, as excessively increasing Tor latency is discouraged by Tor developers [52]. 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 [83], 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

[1] James K Holland and Se Eun Oh are the corresponding authors.

Proceedings on Privacy Enhancing Technologies 2024(1), 98–115

https://doi.org/10.56553/popets-2024-0007

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 re-train 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 [20, 43, 60]. 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. Inspired by generative adversarial networks (GANs) [21], 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 [26]. 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 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.

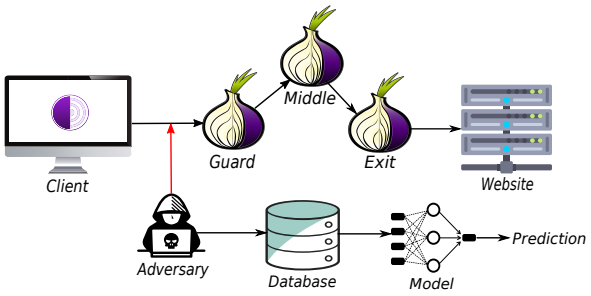


Figure 1: Attacker position and model creation for a WF attack on Tor. The adversary creates a database of download traces for multiple sites and uses them to train a model, which is applied to observed traces between a client and guard relay to predict the downloaded site.

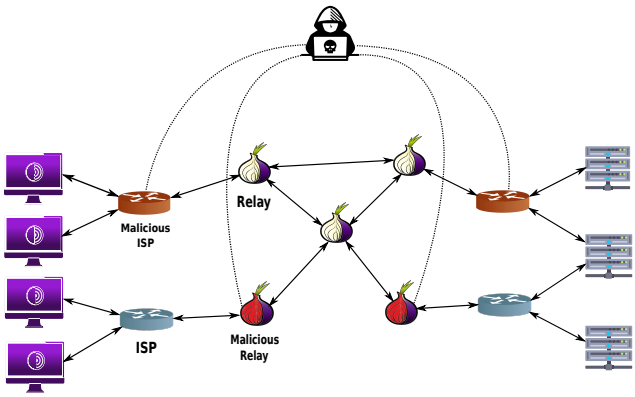


Figure 2: Attacker positioning for a FC attack on Tor. The attacker must collect traffic at both the entry and exit sides of the network, though there are multiple possible vantage points, including the ISPs and malicious relays.

2 BACKGROUND AND RELATED WORK

2.1 Website Fingerprinting

2.1.1 Threat Model. In the WF attack model, shown in Figure 1, we assume that the adversary is a *passive* eavesdropper *local* to the client. Accordingly, this adversary can observe traffic between the Tor client and the entry node but does not modify that traffic. We also assume that this adversary knows the IP address of the user and is trying to determine which website the user is visiting.

2.1.2 Attacks. While data sent through Tor is encrypted, traffic metadata such as packet timing and direction may indicate the destination of a user's traffic. Hintz et al. were the first to demonstrate the risk of WF attacks using file transfer sizes to distinguish web pages [25]. 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 [11, 51]. Later improvements came in the form of dataset collection and feature engineering [76],

the use of more effective machine learning techniques such as k-nearest neighbors and random forests [22, 75], and the use of the cumulative representation of the traces [50].

More recent techniques leveraged deep neural networks to substantially increase WF attack performance [47, 61, 69, 70]. Most importantly, Deep Fingerprinting by Sirinam et al. [69] managed to defeat the WTF-PAD defense while pushing closed-world accuracy above 98%. Then, Rahman et al. [61] added timing information to the Deep Fingerprinting architecture to develop the Tik-Tok attack, further increasing performance with especially strong results against the Walkie-Talkie defense [61]. Due to the success of the convolutional neural network architecture used by Deep Fingerprinting, we use it as the basis of DeTorrent's embedder and discriminator, with some modifications described in Appendix A.1. While some recent attacks focused on outside sources of information to reduce false positives [59] or training WF attacks in more challenging and data-limited settings [46, 70], we use the Deep Fingerprinting and Tik-Tok attacks to benchmark our defenses as they represent versatile state-of-the-art attacks that have proven to be effective against WF defenses.

2.1.3 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 [9, 10, 15]. However, these defenses incur prohibitively high latency and bandwidth overheads. Another defense, RegulaTor [27], also aims to regularize traffic to prevent accurate classification. However, it operates with a more reasonable overhead by sending real traffic 'surges' in standardized bursts, decreasing the amount of information leaked. Because these defenses delay traffic, they are not used for comparison in this paper.

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 [82], Decoy pages [51], Supersequence [75], and Glove [45]. Additionally, the Walkie-Talkie defense [78] modifies the browser to operate in half-duplex mode while modifying burst sequences. These defenses have either been broken by WF attacks or require additional infrastructure or information about traffic associated with other web pages, so we do not evaluate them in this paper.

Additional latency overhead is harmful to the user experience, so some defenses have taken the approach of avoiding traffic delays. WTF-PAD [33] 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 [18], 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 [58] defenses operate using the Tor circuit padding framework [52], 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. Because these defenses have reasonable overhead with no added latency, we directly compare them to DeTorrent.

Some defenses use adversarial techniques to defend traffic. Mockingbird [60] 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. Shan et al. [66] designed a defense, Dolos, that precomputed input-agnostic 'adversarial patches' that could be used to disrupt deep learning-based attacks in real-time. Nasr et al. [43] proposed a technique using blind adversarial perturbations (BAP) to defeat DNN-based traffic analysis. Essentially, the approach trained DNN adversarial traffic generators that aimed to reduce the performance of WF and FC attacks. While BAP is quite effective against a WF or FC adversary using a static DNN model, we assume in this paper that the attacker will be able to retrain their models once traffic is defended. Accordingly, our BAP implementation in this paper will likely appear less effective than expected. Still, BAP's use of a DNN traffic perturbation generator was influential to DeTorrent's development. Surakav [20] uses a Wasserstein GAN [4] 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. While Surakav appears effective, it delays packets while shaping the bursts, so it is not included in our analysis.

The TrafficSliver [13] and Multihoming [23] 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. While these defenses appear effective, they do not guard against all attacker types considered in this paper. For example, TrafficSliver does not defend against local attackers who can see outgoing traffic before it is split. Furthermore, Multihoming requires additional infrastructure to function. As a result, traffic-splitting defenses are not used for comparison in this paper.

2.2 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 [14, 37, 41, 49, 62, 79]. While initial attack formulations included techniques such as 'packet counting' [62], later approaches turned to statistical correlation metrics such as the cross-correlation between binned packet volumes [37, 68], distance functions using mutual information and frequency analysis [84], or Spearman's rank correlation coefficient [73]. Defensive techniques proposed in response include defensive dropping [37], adaptive padding [68], and constant-rate sending [37]. However, constant-rate sending is costly in terms of bandwidth and latency overhead, while defensive dropping has been proven ineffective [68]. Still, we test the performance of an adaptive padding approach in the flow correlation setting later in this paper.

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 [14]. Later work introduced the concept of an autonomous system (AS) level adversary [17]. While these adversaries have a more limited view of the network, it is still quite possible for multiple nodes to be part of the same administrative domain. This risk is then exacerbated by the existence of Internet exchange points (IXPs), which allow the exchange of traffic between autonomous systems, as well as the possibility of IXP coalitions [31]. AS-level adversaries may also be able to increase their chances of intercepting traffic through active attacks, such as BGP hijacks and BGP interception [73, 74]. 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 [2, 6, 16, 35, 45, 72].

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 [8]. 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 [5]. See Figure 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 overcome by recent work by Nasr et al. and Oh et al. with their DeepCorr and DeepCoFFEA attacks [42, 48]. 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. Given that DeepCoFFEA is a state-of-the-art flow correlation attack, we use it in this paper to test the performance of various potential flow correlation defenses. We also implemented the flow correlation defense Decaf, which was presented alongside the DeepCoFFEA attack. It operates by recording windows of traffic from other traffic flows, and then using the packet timing from a randomly selected window to defend a window of real traffic. This way, the window of real traffic is more difficult to embed accurately.

2.3 LSTMs

Recurrent neural networks 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, LSTMs 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. As a result, an LSTM can make a series of decisions regarding inserting dummy packets into a traffic flow *while the traffic is being sent.*

3 DETORRENT DESIGN

3.1 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 [18], Surakav [20], and Mockingbird [60].

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.

3.2 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 [69].

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 [27]. Thus, the upload padding is handled based on observed download traffic, as discussed in Section 3.5. 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' [64], 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 [50], 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 [18, 20, 27]. 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 [67].

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 3.5. 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.

3.3 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 [48] 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.

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

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.

3.4 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 3. 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,

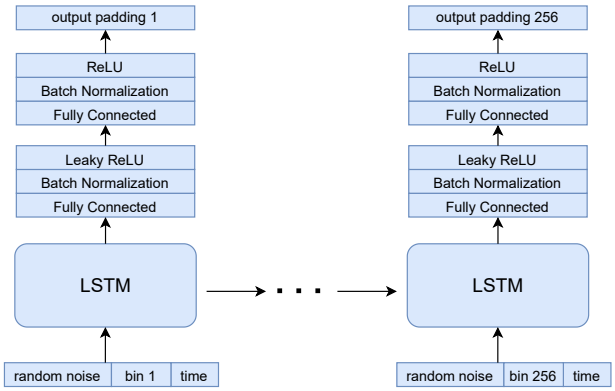


Figure 3: Unrolled LSTM generator outputting the dummy traffic volume at each step. Note that the volume in the previous bin, the time since the start of the defense, and random noise are used as LSTM input at each step.

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 3.2, 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.

3.5 Packet Timing

3.5.1 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 long gaps between bursts [12, 30], so we choose inter-packet delays from the exponential distribution, as done in some previous works [28, 44]. 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.

3.5.2 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. For a more in-depth explanation of our method of estimating traffic rate, see the appendix Section B.

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.

4 DETORRENT TRAINING

4.1 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 1 and illustrated in Figure 4, 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 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

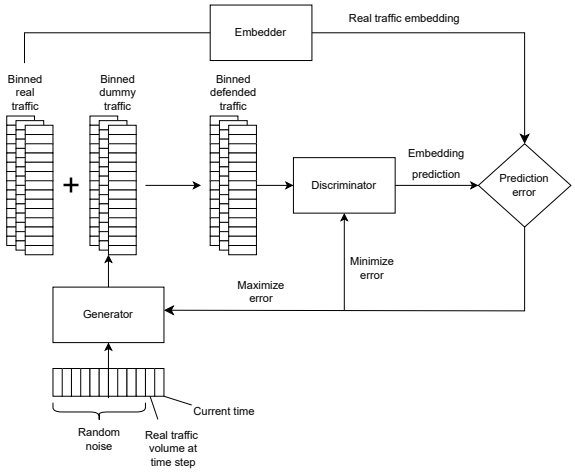


Figure 4: WF-DeTorrent training, where the generator minimizes the discriminator's ability to provide accurate embedding predictions. The binned real traffic and the binned dummy traffic are added to one another to create the defended trace, and the embeddings are computed using a pre-trained embedder.

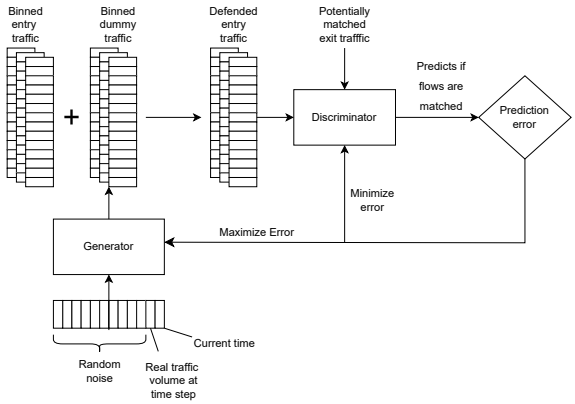


Figure 5: FC-DeTorrent Training, where the generator is trained to reduce the discriminator's ability to predict which flows are matched. The discriminator is trained to determine which flow pairs are associated with the same connection, and the generator is trained to defend the traffic to prevent this.

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 3.3.

4.2 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 5. 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.

As described in Algorithm 2, 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.

5 EXPERIMENT DETAILS

5.1 Datasets

The BigEnough dataset, referred to here as 'BE,' was collected for a Systemization of Knowledge paper on website fingerprinting defenses [40] by Matthews et al. The BE collection methodology was based on that of Pulls for the GoodEnough dataset [58] 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 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 [29] 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

Algorithm 1 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

Algorithm 2 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

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 [69], 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. [76]. 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 [48]. Their dataset collection methodology was based on that of the DeepCorr technique by Nasr et al. [42], 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 [3] 100 times over two weeks in September 2022. To collect the dataset, we used a fork of Tor Browser Crawler [80] 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 5.4, 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.'

5.2 WF Attack Details

We used open-source code associated with WTF-PAD [34], Interspace [57], Spring [57], FRONT [19], and BAP [71] 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 [58] 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 1 and Table 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 [81]. 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 [57] 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.

5.3 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 [43]. 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 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 [42] 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 5.2. We again refrain from doing hyperparameter optimization to tune FC-DeTorrent's training and implementation.

5.4 Real-world Implementation Details

To demonstrate the practicality of the DeTorrent defense, we implement it as a pluggable transport [56]. Pluggable transports (PTs) transform traffic between the client and a Tor bridge [54], 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 [32], which extends the Obfsproxy PT [53] 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 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.

6 WEBSITE FINGERPRINTING RESULTS

6.1 Simulated Defense Results

Defense performance on the BE dataset is shown in Table 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

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 1: Defense Overheads and Parameters on DF

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 [58]
Interspace	96.7%	see [58]
FRONT	95.0%	$N_s = N_c = 4000, W_{min} = 1, W_{max} = 14$
WF-DeTorrent	98.9%	$N_{download} = 3000$

Table 2: Defense Overheads and Parameters on BE

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 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 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 5: Defense Overheads on DCF

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 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 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

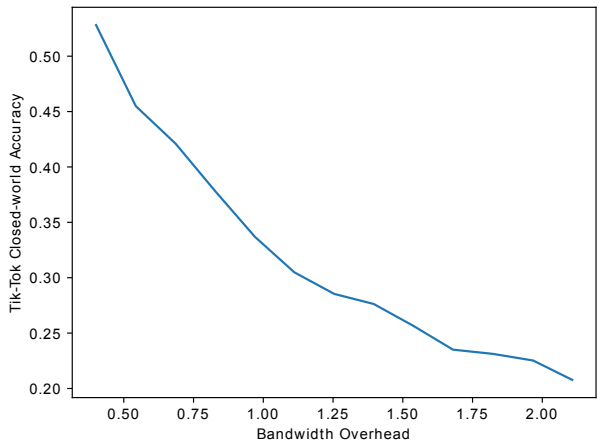


Figure 6: DeTorrent Overhead vs. Performance

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.

6.2 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 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%.

6.3 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, 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.

6.4 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. [40]. 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).

6.5 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

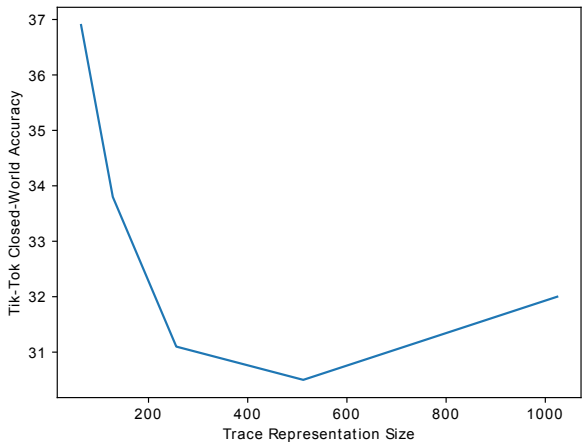


Figure 7: DeTorrent Performance vs. Representation Length

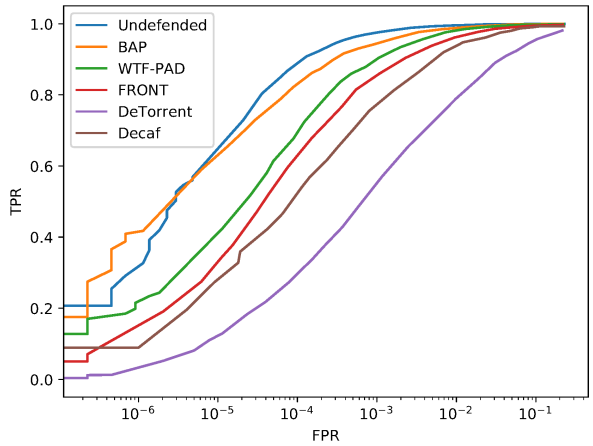


Figure 8: Flow correlation defense performance against DeepCoFFEA. Note that lower TPR implies a better defense.

websites, we find that defense performance remains high, reducing Tik-Tok to a 21.5% accuracy with an 88.8% bandwidth overhead.

7 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 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 8.

In terms of reducing DeepCoFFEA's performance, FC-DeTorrent far outperforms the compared defenses, as shown in Figure 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.

8 DISCUSSION

8.1 Transferability

8.1.1 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.

8.1.2 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.

8.2 Overhead

While DeTorrent doesn't add any latency of its own, past work by Witwer et al. on network-wide Tor simulation [81] 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 1 and 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.

8.3 Implementation Framework

While the PT system is convenient due to its relative flexibility in terms of supporting various traffic analysis defenses [56], 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 [52], 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.

9 CONCLUSION AND FUTURE WORK

In this paper, we present the padding-only traffic analysis defense DeTorrent, which uses competing neural networks to generate defense strategies resistant to adversarial retraining. To train the defense generator to make real-time decisions, we implement it as an LSTM that takes into account live traffic patterns. We describe how we represent the trace as a tensor and use an embedder to create a useful notion of distance between traces, allowing for more effective training in the WF setting.

To test WF-DeTorrent's defense performance, we compare it to other padding-only defenses scaled to use similar bandwidth overhead. In the closed-world WF attack on the BE dataset, we find that it outperforms comparable defenses with a 10.5% larger accuracy reduction than the next best defense. It also reduces the accuracy by 61.5% compared to the undefended traffic, indicating its ability to defend against state-of-the-art WF attacks. Similarly, we find that FC-DeTorrent reduces the true positive rate of the flow correlation attack DeepCoFFEA at the 10^{-4} false positive rate threshold to just .30 compared to about .54 for the next best defense. Furthermore, we test the real-world DeTorrent implementation against live traffic and find that it achieves similar success, reducing Tik-Tok attack performance against the PT dataset to just 21.5%. Overall, the results show that DeTorrent can both defend against traffic analysis attacks and be deployed practically. Future work may include experimenting with alternate neural network architectures or testing how delaying packets improves defense performance.

10 APPENDICES

ACKNOWLEDGMENTS

We'd like to thank Nate Matthews for sharing the BigEnough dataset and for his work on tor-crawler-pluggable, which we used to collect the PT dataset. This work was funded by a 3M fellowship and NSF grants 1814753 and 1815757. It was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. RS-2022-00166669, RS-2023-00222385) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT)

(No. RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University)).

This project was partially sponsored by the National Security Agency under Grant H98230-22-1-0302. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation herein. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Security Agency.

REFERENCES

[1] Kota Abe and Shigeki Goto. 2016. Fingerprinting Attack on Tor Anonymity using Deep Learning. *Proceedings of the Asia-Pacific Advanced Network* 42, 0 (2016), 15–20.

[2] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. 2014. LASTor: A Low-Latency AS-Aware Tor Client. *IEEE/ACM Trans. Netw.* 22, 6 (dec 2014), 1742–1755. https://doi.org/10.1109/TNET.2013.2291242

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

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 214–223. https://proceedings.mlr.press/v70/arjovsky17a.html

[5] arma. 2011. Research problem: better guard rotation parameters. https://blog.torproject.org/research-problem-better-guard-rotation-parameters/ Accessed: 2022-7-14.

[6] Armon Barton and Matthew Wright. 2016. DeNASA: Destination-Naive AS-Awareness in Anonymous Communications. *Proceedings on Privacy Enhancing Technologies* 2016 (02 2016). https://doi.org/10.1515/popets-2016-0044

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

[8] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. 2007. Denial of Service or Denial of Security? How Attacks on Reliability can Compromise Anonymity. In *Proceedings of CCS 2007*.

[9] Xiang Cai, Rishab Nithyanand, and Rob Johnson. 2014. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (Scottsdale, Arizona, USA) *(WPES '14)*. Association for Computing Machinery, New York, NY, USA, 121–130. https://doi.org/10.1145/2665943.2665949

[10] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the ACM Conference on Computer and Communications Security*. Association for Computing Machinery, 227–238. https://doi.org/10.1145/2660267.2660362

[11] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 605–616. https://doi.org/10.1145/2382196.2382260

[12] M.E. Crovella and A. Bestavros. 1997. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking* 5, 6 (1997), 835–846. https://doi.org/10.1109/90.650143

[13] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. 2020. TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1971–1985. https://doi.org/10.1145/3372297.3423351

[14] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium (USENIX Security 04)*. USENIX Association, San Diego, CA. https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router

[15] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *2012 IEEE Symposium on Security and Privacy*. 332–346. https://doi.org/10.1109/SP.2012.28

[16] Matthew Edman and Paul Syverson. 2009. As-Awareness in Tor Path Selection. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '09)*. Association for Computing Machinery, New York, NY, USA, 380–389. https://doi.org/10.1145/1653662.1653708

[17] Nick Feamster and Roger Dingledine. 2004. Location Diversity in Anonymity Networks. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society* (Washington DC, USA) *(WPES '04)*. Association for Computing Machinery, New York, NY, USA, 66–76. https://doi.org/10.1145/1029179.1029199

[18] Jiajun Gong and Tao Wang. 2020. Zero-delay Lightweight Defenses against Website Fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 717–734. https://www.usenix.org/conference/usenixsecurity20/presentation/gong

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

[20] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. 2022. Surakav: Generating Realistic Traces for a Strong Website Fingerprinting Defense. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1558–1573. https://doi.org/10.1109/SP46214.2022.9833722

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[22] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 1187–1203. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes

[23] Sébastien Henri, Gines Garcia-Aviles, P. Serrano, A. Banchs, and P. Thiran. 2020. Protecting against Website Fingerprinting with Multihoming. *Proceedings on Privacy Enhancing Technologies* 2020 (2020), 89 – 110.

[24] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security* (Chicago, Illinois, USA) *(CCSW '09)*. Association for Computing Machinery, New York, NY, USA, 31–42. https://doi.org/10.1145/1655008.1655013

[25] Andrew Hintz. 2002. Fingerprinting Websites Using Traffic Analysis. In *Proceedings of the 2nd International Conference on Privacy Enhancing Technologies* (San Francisco, CA, USA) *(PET'02)*. Springer-Verlag, Berlin, Heidelberg, 171–178.

[26] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (11 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735 arXiv:https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf

[27] James K Holland and Nicholas Hopper. 2022. RegulaTor: A Straightforward Website Fingerprinting Defense. *Proceedings on Privacy Enhancing Technologies* 2022 (2022). https://petsymposium.org/popets/2022/popets-2022-0049.pdf

[28] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. 2012. Non-Blind Watermarking of Network Flows. *IEEE/ACM Transactions on Networking* 22 (03 2012). https://doi.org/10.1109/TNET.2013.2272740

[29] Open PageRank Initiative. 2022. What is Open PageRank? https://www.domcop.com/openpagerank/what-is-openpagerank. Accessed: 2022-08-01.

[30] Hao Jiang and Constantinos Dovrolis. 2005. Why is the Internet Traffic Bursty in Short Time Scales? *SIGMETRICS Perform. Eval. Rev.* 33, 1 (jun 2005), 241–252. https://doi.org/10.1145/1071690.1064240

[31] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 337–348. https://doi.org/10.1145/2508859.2516651

[32] Marc Juarez. 2015. WFPadTools. https://github.com/mjuarezm/wfpadtools. Accessed: 2021-5-21.

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

[34] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright. 2022. WTF-PAD. https://github.com/wtfpad/wtfpad Accessed: 2022-7-14.

[35] Joshua Juen, Anupam Das, Aaron Johnson, Nikita Borisov, and Matthew Caesar. 2014. Defending Tor from Network Adversaries: A Case Study of Network Path Prediction. *CoRR* abs/1410.1823 (2014). arXiv:1410.1823 http://arxiv.org/abs/1410.1823

[36] Ilmari Karonen. 2014. Estimating rate of occurrence of an event with exponential smoothing and irregular events. https://stackoverflow.com/questions/23615974/estimating-rate-of-occurrence-of-an-event-with-exponential-smoothing-and-irregul?noredirect=1&lq=1 Accessed: 2022-8-13.

[37] Brian N. Levine, Michael K. Reiter, Chenxi Wang, and Matthew Wright. 2004. Timing Attacks in Low-Latency Mix Systems. In *Financial Cryptography*, Ari Juels (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 251–265.

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

[39] Akshaya Mani, T. Wilson-Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. 2018. Understanding Tor Usage with Privacy-Preserving Measurement. In *Proceedings of the Internet Measurement Conference 2018* (Boston, MA, USA) *(IMC '18)*. Association for Computing Machinery, New York, NY, USA, 175–187. https://doi.org/10.1145/3278532.3278549

[40] Nate Matthews, James Holland, Se Eun Oh, Mohammad Saidur Rahman, Matthew Wright, and Nicholas Hopper. 2023. SoK: A Critical Evaluation of Efficient Website Fingerprinting Defenses. In *2023 IEEE Symposium on Security and Privacy*. IEEE.

[41] S.J. Murdoch and G. Danezis. 2005. Low-cost traffic analysis of Tor. In *2005 IEEE Symposium on Security and Privacy (S&P'05)*. 183–195. https://doi.org/10.1109/SP.2005.12

[42] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2018. DeepCorr. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM. https://doi.org/10.1145/3243734.3243824

[43] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2021. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2705–2722. https://www.usenix.org/conference/usenixsecurity21/presentation/nasr

[44] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. 2017. Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2053–2069. https://doi.org/10.1145/3133956.3134074

[45] Rishab Nithyanand, Xiang Cai, and Rob Johnson. 2014. Glove: A bespoke website fingerprinting defense. In *Proceedings of the ACM Conference on Computer and Communications Security*. Association for Computing Machinery, 131–134. https://doi.org/10.1145/2665943.2665950

[46] Se Oh, Nate Mathews, Mohammad Saidur Rahman, Matthew Wright, and Nicholas Hopper. 2020. GANDaLF: GAN for Data-Limited Fingerprinting. *Proceedings on Privacy Enhancing Technologies* 2021. https://doi.org/10.2478/popets-2021-0029

[47] Se Eun Oh, Saikrishna Sunkam, and Nicholas Hopper. 2019. p1-FP: Extraction, Classification, and Prediction of Website Fingerprints with Deep Learning. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 191–209. https://doi.org/10.2478/popets-2019-0043 arXiv:arXiv:1711.03656v2

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

[49] L. Overlier and P. Syverson. 2006. Locating hidden servers. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*. 15 pp.–114. https://doi.org/10.1109/SP.2006.24

[50] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. 2017. Website Fingerprinting at Internet Scale. February (2017), 21–24. https://doi.org/10.14722/ndss.2016.23477

[51] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society* (Chicago, Illinois, USA) *(WPES '11)*. Association for Computing Machinery, New York, NY, USA, 103–114. https://doi.org/10.1145/2046556.2046570

[52] Mike Perry and George Kadianakis. 2021. Circuit Padding Developer Documentation. https://github.com/torproject/tor/blob/master/doc/HACKING/CircuitPaddingDevelopment.md. Accessed: 2022-09-17.

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

[54] Tor Project. 2022. Get Bridges for Tor. https://bridges.torproject.org/ Accessed: 2022-7-28.

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

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

[57] Tobias Pulls. 2019. Tor Circuit Padding Simulator. https://github.com/pylls/circpad-sim Accessed: 2022-7-14.

[58] Tobias Pulls. 2022. Towards Effective and Efficient Padding Machines for Tor. (2022). https://arxiv.org/pdf/2011.13471.pdf

[59] Tobias Pulls and Rasmus Dahlberg. 2020. Website Fingerprinting with Website Oracles. *Proceedings on Privacy Enhancing Technologies* 2020 (01 2020), 235–255. https://doi.org/10.2478/popets-2020-0013

[60] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew K. Wright. 2021. Mockingbird: Defending Against Deep-Learning-Based Website Fingerprinting Attacks With Adversarial Traces. *IEEE Transactions on Information Forensics and Security* 16 (2021), 1594–1609.

[61] Mohammad Saidur Rahman, Payap Sirinam, Nate Matthews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks. *Proceedings of Privacy Enhancing Technologies* (2020). https://petsymposium.org/popets/2020/popets-2020-0043.pdf

[62] Jean-François Raymond. 2001. *Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 10–29. https://doi.org/10.1007/3-540-44702-4_2

[63] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. 1998. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications* 16, 4 (1998), 482–494. https://doi.org/10.1109/49.668972

[64] NumPy Reference. 2023. numpy.geomspace. https://numpy.org/doc/stable/reference/generated/numpy.geomspace.html Accessed: 2023-7-07.

[65] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. (2018). https://doi.org/10.14722/ndss.2018.23105

[66] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y. Zhao. 2021. Patch-Based Defenses against Web Fingerprinting Attacks. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security* (Virtual Event, Republic of Korea) *(AISec '21)*. Association for Computing Machinery, New York, NY, USA, 97–109. https://doi.org/10.1145/3474369.3486875

[67] Meng Shen, Kexin Ji, Zhenbo Gao, Qi Li, Liehuang Zhu, and Ke Xu. 2023. Subverting Website Fingerprinting Defenses with Robust Traffic Representation. In *32th USENIX Security Symposium (USENIX Security 23)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity23/presentation/shenmeng

[68] Vitaly Shmatikov and Ming-Hsiu Wang. 2006. Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses. In *Computer Security – ESORICS 2006*, Dieter Gollmann, Jan Meier, and Andrei Sabelfeld (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 18–33.

[69] Payap Sirinam, Marc Juarez, Mohsen Imani, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. *Proceedings of the ACM Conference on Computer and Communications Security* (2018), 1928–1943. https://doi.org/10.1145/3243734.3243768

[70] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. 2019. 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* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 1131–1148. https://doi.org/10.1145/3319535.3354217

[71] SPIN-UMass. 2022. BLANKET. https://github.com/SPIN-UMass/BLANKET Accessed: 2022-7-14.

[72] Y. Sun, A. Edmundson, N. Feamster, M. Chiang, and P. Mittal. 2017. Counter-RAPTOR: Safeguarding Tor Against Active Routing Attacks. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 977–992. https://doi.org/10.1109/SP.2017.34

[73] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. 2015. RAPTOR: Routing Attacks on Privacy in Tor. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 271–286. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sun

[74] Henry Tan, Michael E. Sherr, and Wenchao Zhou. 2016. Data-plane Defenses against Routing Attacks on Tor. *Proceedings on Privacy Enhancing Technologies* 2016 (2016), 276 – 293.

[75] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. *23rd USENIX Security Symposium (USENIX Security 14)* (2014), 143–157. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/wang_tao

[76] Tao Wang and Ian Goldberg. 2013. Improved website fingerprinting on Tor. *Proceedings of the ACM Conference on Computer and Communications Security* (2013), 201–212. https://doi.org/10.1145/2517840.2517851

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

[78] Tao Wang and Ian Goldberg. 2017. Walkie-Talkie: An Efficient Defense against Passive Website Fingerprinting Attacks. In *Proceedings of the 26th USENIX Conference on Security Symposium* (Vancouver, BC, Canada) *(SEC'17)*. USENIX Association, USA, 1375–1390.

[79] Xinyuan Wang, Douglas S. Reeves, and S. Felix Wu. 2002. Inter-Packet Delay Based Correlation for Tracing Encrypted Connections through Stepping Stones. In *Computer Security — ESORICS 2002*, Dieter Gollmann, Günther Karjoth, and Michael Waidner (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 244–263.

[80] webfp. 2017. Tor Browser Crawler. https://github.com/webfp/tor-browser-crawler. Accessed: 2021-11-17.

[81] Ethan Witwer, James K. Holland, and Nicholas Hopper. 2022. Padding-Only Defenses Add Delay in Tor. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society* (Los Angeles, CA, USA) *(WPES'22)*. Association for Computing Machinery, New York, NY, USA, 29–33. https://doi.org/10.1145/3559613.3563207

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

[83] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial Examples: Attacks and Defenses for Deep Learning. *IEEE Transactions on Neural Networks and Learning Systems* 30, 9 (2019), 2805–2824. https://doi.org/10.1109/TNNLS.2018.2886017

[84] Ye Zhu, Xinwen Fu, Bryan Graham, Riccardo Bettati, and Wei Zhao. 2005. On Flow Correlation Attacks and Countermeasures in Mix Networks. In *Privacy*

Enhancing Technologies, David Martin and Andrei Serjantov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 207–225.

A IMPLEMENTATION DETAILS

A.1 Model Architecture

The discriminator and embedder in the WF setting use the architecture shown in Figure 9. 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 10. 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 3, 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.

A.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.

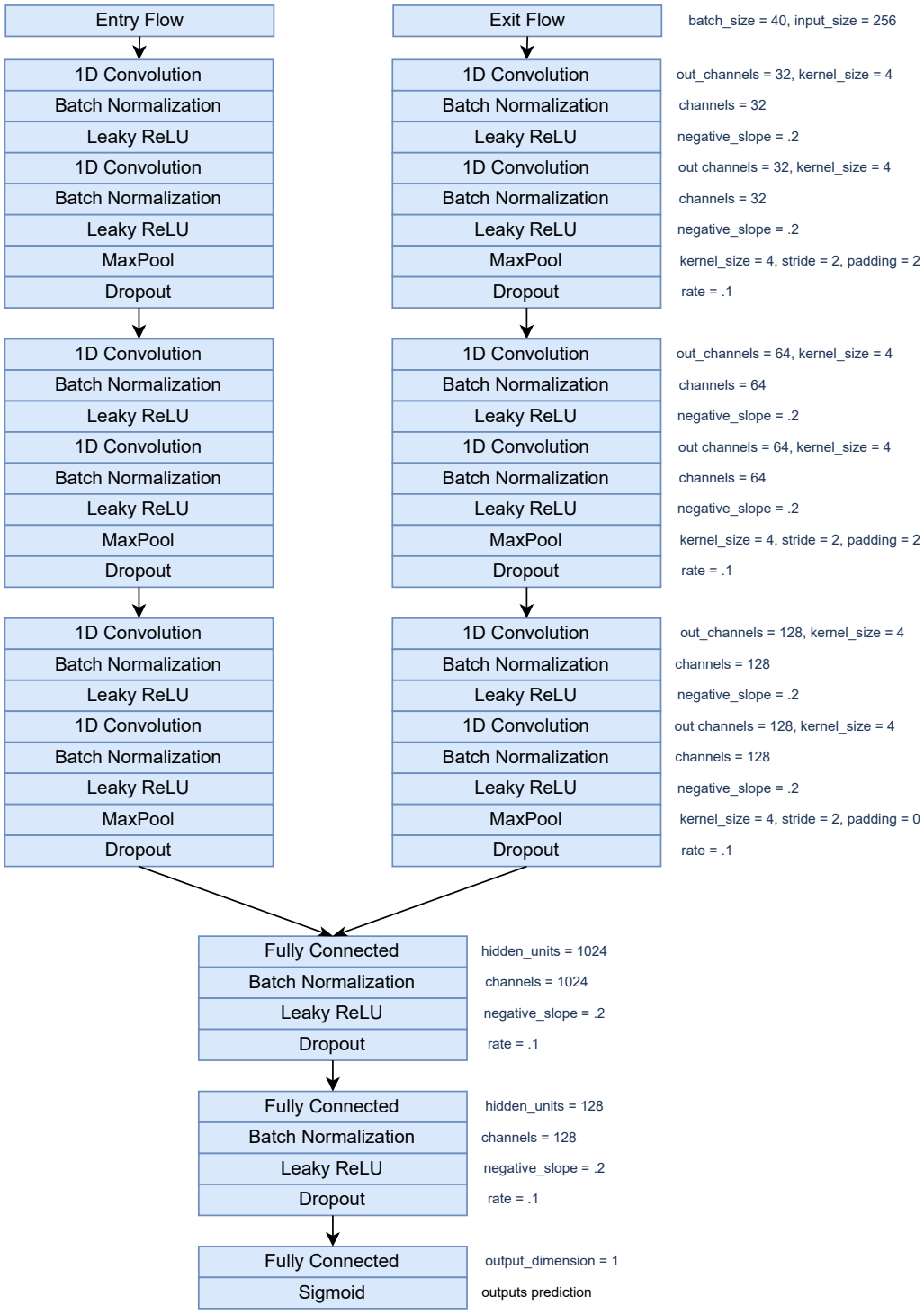
One implementation challenge for DeTorrent is distributing the models. Because the upload padding is quite simple, only the generator needs to be distributed to bridges or relays. Our proposal for model distribution is similar to the one presented for the Surakav defense [20], which is that the semi-trusted 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.

Figure 9

B 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

Figure 10


Entry Flow
1D Convolution
Batch Normalization
Leaky ReLU
1D Convolution
Batch Normalization
Leaky ReLU
MaxPool
Dropout
1D Convolution
Batch Normalization
Leaky ReLU
1D Convolution
Batch Normalization
Leaky ReLU
MaxPool
Dropout
1D Convolution
Batch Normalization
Leaky ReLU
1D Convolution
Batch Normalization
Leaky ReLU
MaxPool
Dropout
Exit Flow
batch_size = 40, input_size = 256
1D Convolution
Batch Normalization
Leaky ReLU
1D Convolution
Batch Normalization
Leaky ReLU
MaxPool
Dropout
1D Convolution
Batch Normalization
Leaky ReLU
1D Convolution
Batch Normalization
Leaky ReLU
MaxPool
Dropout
1D Convolution
Batch Normalization
Leaky ReLU
1D Convolution
Batch Normalization
Leaky ReLU
MaxPool
Dropout
out_channels = 32, kernel_size = 4
channels = 32
negative_slope = .2
out channels = 32, kernel_size = 4
channels = 32
negative_slope = .2
kernel_size = 4, stride = 2, padding = 2
rate = .1
out_channels = 64, kernel_size = 4
channels = 64
negative_slope = .2
out channels = 64, kernel_size = 4
channels = 64
negative_slope = .2
kernel_size = 4, stride = 2, padding = 2
rate = .1
out_channels = 128, kernel_size = 4
channels = 128
negative_slope = .2
out channels = 128, kernel_size = 4
channels = 128
negative_slope = .2
kernel_size = 4, stride = 2, padding = 0
rate = .1
Fully Connected
Batch Normalization
Leaky ReLU
Dropout
Fully Connected
Batch Normalization
Leaky ReLU
Dropout
Fully Connected
Sigmoid
hidden_units = 1024
channels = 1024
negative_slope = .2
rate = .1
hidden_units = 128
channels = 128
negative_slope = .2
rate = .1
output_dimension = 1
outputs prediction

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 [36] 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.