WTF-PAD: Toward an Efficient Website Fingerprinting Defense

Marc Juarez¹, Mohsen Imani², Mike Perry³, Claudia Diaz¹, Matthew Wright²

¹KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium
{marc.juarez,claudia.diaz}@esat.kuleuven.be
²The University of Texas at Arlington, TX, USA
mwright@cse.uta.edu, mohsen.imani@mavs.uta.edu
³The Tor Project, https://torproject.org
mikeperry@torproject.org

Abstract

Website Fingerprinting attacks enable a passive eavesdropper to recover the user’s otherwise anonymized web browsing activity by matching the observed traffic with prerecorded web page traffic templates. The defenses that have been proposed to counter these attacks are impractical for deployment in real-world systems due to their high cost in terms of added delay and bandwidth overhead. Further, these defenses have been designed to counter attacks that, despite their high success rates, have been criticized for assuming unrealistic attack conditions in the evaluation setting. In this paper, we propose a novel, lightweight defense based on Adaptive Padding that provides a sufficient level of security against website fingerprinting, particularly in realistic evaluation settings. In a closed-world setting, this defense reduces the accuracy of the state-of-the-art attack from 91% to 20%, while introducing zero latency overhead and less than 60% bandwidth overhead. In an open-world setting, the attack precision is just 1% and drops further as the number of sites grows. For the implementation and evaluation of the defense, we have developed a tool for evaluating the traffic analysis resistance properties of Tor Pluggable Transports that we hope will contribute to future research on traffic analysis.

1 Introduction

Website Fingerprinting (WF) is a type of traffic analysis that allows the attacker to recover the browsing history of a client. WF is known to be effective in a wide range of scenarios, ranging from HTTPS connections [18], SSH tunnels [11], one-hop proxies [12], VPNs [21] and even anonymous communication systems such as Tor [5].

The success of WF against Tor is particularly problematic. Tor is one of the largest deployed systems for anonymously browsing the Web [22]. Since it offers stronger security than one-hop proxies, it is meant to protect against attacks like WF that require only a local eavesdropper or a compromised guard node. However, State-of-the-art WF attacks achieve more than 90% accuracy against Tor [5, 26, 25], thus breaking the anonymity properties that Tor aims to provide.

To counter these attacks, a broad range of defenses has been proposed (see Section 2). The key building block of most of these defenses is link padding. Link padding adds varying amounts of delays and dummy messages to the packet flows to conceal patterns in network traffic. Given that bandwidth and latency increases come at a cost to usability and deployability, these defenses must strive for a trade-off between security and performance overheads. Unfortunately, the current trade-off in state-of-the-art link-padding defenses is not acceptable for use in Tor. They increase latency so that pages load on average between two and four times slower and impose bandwidth overheads between 40% and 350%.

We note that any delays introduced by a defense are especially a concern for low-latency systems, as they have a direct impact on the usability of the system in interactive applications. Moderate bandwidth overheads may also impact the user experience but the load factor needs to increase substantially before being noticeable by users. Moreover, the Tor network has spare bandwidth on its entry and middle routers (the first two out of three hops in a Tor circuit), making it possible to afford a defense that consumes a moderate amount of bandwidth. In this work, we thus explore the design space of effective link-padding defenses with minimal (ideally zero) latency overhead and modest bandwidth overhead.

The contributions of the following sections are:

An analysis of the suitability of existing WF defenses for deployment in Tor. In Section 2, we give a background of existing attacks and defenses, and we discuss the suitability of the defenses for an implementation in Tor. Section 3 provides a description of the system model considered in this paper.
A framework for the systematic implementation and evaluation of link-padding WF defenses. In Section 4 we give a description of the framework and describe its main components. We will make the source code of the framework and the data used in this study available to other researchers upon publication.

A lightweight defense against WF attacks. We have adapted Adaptive Padding to combat WF in Tor and dubbed this new defense Website Traffic Fingerprinting Protection with Adaptive Defense (WTF-PAD). Section 5 gives its specification, and Section 6 presents an evaluation and a comparison of WTF-PAD with the existing WF defenses. We find that WTF-PAD is effective and has reasonable overheads for a system like Tor.

An evaluation of the defense in realistic scenarios. Prior work has shown that the accuracy of the WF attack decreases significantly when certain assumptions about the setting or user behavior do not hold [14], but to the best of our knowledge this is the first study that evaluates the effectiveness of a WF defense in these scenarios. In Section 7, we show the results for two realistic scenarios: (i) open-world, in which the attacker monitors a small set of web pages and, (ii) multi-tab, where the users browse the pages using multiple tabs. We show that for these scenarios, the defense substantially reduces the accuracy of the state-of-the-art WF attack.

2 Website Fingerprinting (WF)

In this section we provide the necessary background on WF attacks and defenses to understand and put in context the contributions of the rest of the paper.

2.1 Attacks

Though several studies show how to identify specific pages within a single website [6], the WF attack that we address considers an adversary whose objective is to identify visits to website home pages within the whole space of websites. Early works on this problem [12, 21] assumed a user model that could only access a small set of pages—an assumption that is unlikely to be met in practice. This assumption is known as the closed-world assumption, and it overly simplifies the problem to the point of being irrelevant to most real-world settings. In contrast, the more realistic open-world assumption allows the user to visit any page in the Web.

With the first WF attack against Tor, Herrmann et al. got 3% accuracy with a Naive Bayes classifier [11] in a closed world and without any WF countermeasures. Despite the low success rate of this first attempt, the attack was revisited with better classification models and more refined feature sets [19], and state-of-the-art attacks attain over 90% accuracy [5, 26, 25]. Wang and Goldberg report that their attack using the k-NN classifier even offers robustness in an open-world setting with WF countermeasures in place [25].

WF attacks on Tor are a serious threat to Tor’s security, as the adversary only needs the ability to eavesdrop on the connection between the client and the guard node, such as compromising the user’s wireless router or cable/DSL modem, eavesdropping on the wireless connection, having access to the user’s ISP, volunteering to run a malicious guard (the user’s entry to the Tor network), or eavesdropping on the guard. With the continuous improvement in WF classifier accuracy over the past few years, this is a pressing concern.

2.2 Defenses

We find in the literature a wide range of WF countermeasures. Of these, most are theoretical designs without a specification for a practical implementation. Only a few have been implemented and evaluated for anonymous communications, and the only one that is currently implemented in Tor does not work as expected. In this section, we review WF defenses proposed in the literature and discuss their suitability for implementation in Tor.

Application-level defenses. These defenses work at the application layer. HTTPS modifies HTTP headers and injects HTTP requests strategically [16], while Randomized Pipelining, a WF countermeasure currently implemented in the Tor Browser, randomizes the pipeline of HTTP requests. Both defenses have been shown to be ineffective in several evaluations [5, 26, 25, 14].

Supersequences and traffic morphing. Recent works have proposed defenses based on generalizing web page traffic traces [25, 3]. They create anonymity sets by clustering pages and defining a centroid for each cluster. Next, they modify traffic traces of pages in a cluster to make them look like the corresponding centroid. This approach aims to optimally reduce the amount of padding needed to confound the attacker’s classifier. These defenses, as well as traffic morphing techniques [27, 15], have the shortcoming that they require a large database of webpage templates that needs to be frequently updated and would be costly to maintain [14].

Low-level defenses. Low-level defenses operate just at the network layer. Dyer et al. evaluated the impact of padding individual packets [9], finding that this is not sufficient to hide coarse-grained features such as bursts in traffic or the total size and load time of the page. Note that Tor cells (onion-encrypted chunks of data) are already padded up to 512 bytes. Dyer et al. also simulated a proof-of-concept countermeasure called BuFLO, which
used constant-rate traffic with fixed-size packets. The authors report excessive bandwidth overheads in return for moderate security. The condition to stop the padding after the transmission ends is critical to adjust the trade-off between overheads and security. BuFLO stops when a page has finished loading and a minimum amount of time has passed. Tuning this minimum time is hard, as the distribution of the download total time in the BuFLO evaluation datasets varies greatly from one page to another.

Tamaraw [4] and CS-BuFLO [5, 2], both attempt to optimize the original design of BuFLO. Instead of setting a minimum duration of padding, Tamaraw stops padding when the total number of transmitted bytes is the multiple of a certain parameter. This approach groups webpages in anonymity sets, with the amount of padding generated being dependent on the webpage’s total size. Given the asymmetry of web browsing traffic, Cai et al. also suggest treating incoming and outgoing traffic independently, using different packet sizes and padding at different rates. Furthermore, the authors sketched CS-BuFLO as a practical version of BuFLO, extended with congestion sensitivity and rate adaptation. Following Tamaraw’s grouping in anonymity sets by page size, they propose either padding up to a power of two, or to a multiple of the power of the amount of transmitted application bytes.

We question the viability of this family of defenses for Tor. Their latency overheads are very high, such as two-to-three times as long to fetch a page, and bandwidth overheads for BuFLO and CS-BuFLO are also over 100%. Also, it is challenging in the modern Web to know when a page has finished loading, as needed in both Tamaraw and CS-BuFLO. Nevertheless, in this paper, we compare our system against these defenses, because they are effective against state-of-the-art attacks and do not require a database of sites to be distributed. This makes them closest to meeting the deployment constraints of Tor.

3 System Model

Here we describe the threat model and the network model that we consider throughout our study. We also describe the requirements for a defense that could realistically be deployed in a system like Tor.

3.1 Adversary Model

As depicted in Figure 1, we assume that the client connects to Tor through a bridge, a volunteer-run proxy to the Tor network. The adversary has access to the communication at some point between the client and the bridge. The adversary is local, meaning that it is unable to observe other parts of the network, and passive, meaning that it observes and records packets but does not modify, delay, drop or inject packets. We also assume that the adversary cannot learn anything about packet payloads due to the use of layered encryption.

The adversary’s objective is to determine whether the client is downloading one of a small set of monitored pages. There is however a much larger set of possible pages that the client could be visiting. This kind of open-world setting fits a surveillance-type of adversary that seeks to deanonymize connections to a blacklist of specific pages. The more distinctive these pages are, the more successful the attack will be. We have also evaluated previously studied open- and closed-world scenarios for the sake of comparison with results in prior work.

3.2 Defense Model

Padding is performed end-to-end between trusted endpoints, with the adversary having access to the padded traces. For this research, we assume the bridge is trusted. This allows us to implement the defense as a Pluggable Transport (PT) [23], avoiding modifications in the Onion Router source code. Note this model is equivalent for a client connecting to the trusted entry guard without a bridge, but in that case the defense would need to be implemented at the guard. To protect against malicious bridges and malicious guards, then the padding should be sent between the client and the middle node.

Most PTs are designed to circumvent deep packet inspection [13], and thus aim to prevent a different, yet complementary, threat than that of WF. We choose to implement our WF framework and countermeasures as PTs for several reasons. First, doing so allows researchers to evaluate these defenses outside the laboratory without introducing excessive overheads in the Tor network. Other advantages of PTs are their modularity, interoperability, and independence from the Tor protocol. Users who are interested in paying the extra costs of a defense could use bridges that run the countermeasure without affecting other users and still be part of the same anonymity set with the rest of the Tor user base.

Any link-padding protocol requires a flag that indicates whether a message must be discarded when it reaches the other end. We added a framing layer to the protocol stack that includes this flag in its header\(^1\). In Figure 2, we...
Figure 2: The network stack of WF defenses implemented as Tor pluggable transports.

can see that this layer sits between TCP and the SOCKS interface used by Tor. Note that this approach introduces an extra layer of encryption in the transmitted messages.

4 Framework

Obfsproxy is the basis of most existing PTs To implement the evaluation framework, we used WFPadTools, a tool that supplies to Obfsproxy the necessary primitives to implement link-padding protocols. We wrapped Obfsproxy classes and minimally extended them to provide writing and reading access to our framework modules. The framework is thus compatible with any transport based on Obfsproxy and can be used to evaluate a broad range of traffic analysis defenses, including new link-padding proposals. The framework comprises the following modules:

Crawler. The crawler allows researchers to create new traffic traces over Tor using a PT with a defense. This is valuable for testing new defenses and for getting fresh data, since websites change frequently.

Replayer. After collecting traces over plain Tor, a researcher may want to apply a defense without having to run a new crawl. She can do this locally using the replayer module (Figure 3). For each packet in a trace, the replayer strips its TCP payload and sends it to the corresponding endpoint (either the client or the relay) to simulate the traces as they were captured on the wire.

Simulator. The simulator applies transformations to an existing TCP stream to reproduce the trace as it would have been captured over the network but, in contrast to the replayer, makes no use of the PT or the network. It is agnostic to artifacts present in real communications and is thus intended for theoretical research.

5 Adaptive Padding

There is a debate on whether probabilistic or deterministic strategies should be deployed to counter WF attacks [25]. Deterministic defenses leak less information than probabilistic ones and provide well-defined guarantees, but they are also more expensive. Given that real-world attacks are not as effective as lab results might suggest, a probabilistic defense may be sufficient to stop the attacks in most common scenarios.

In particular, we believe that Adaptive Padding (AP), proposed by Shmatikov and Wang as a countermeasure against end-to-end traffic analysis [20], can be adapted to protecting against WF due to its generality and flexibility. AP has the defender, in our case the PT client, examine the outgoing traffic pattern and generate dummy messages in a targeted manner to disrupt distinctive features of the patterns — “statistically unlikely” delays between packets. Shmatikov and Wang showed that with 50% bandwidth overhead, the accuracy of end-to-end timing-based traffic analysis is significantly degraded [20].

In the BuFLO family of defenses (BuFLO, CS-BuFLO, Tamaraw), the inter-arrival time between packets is fixed and application data is delayed, if needed, to fit the rigid schedule of packet timings. This adds delays in the common case that multiple real cells are sent all at once, making this family of defenses ill-suited for a system like Tor, as it would significantly harm the user experience. By contrast, Adaptive Padding (AP) does not delay application data; rather, it sends it immediately. This minimal latency overhead makes AP a good candidate for Tor.

Recent WF attacks are significantly different, however, from the end-to-end attacks that AP is designed to counter. Notably, the WF attacker has the advantage of being able to build a dataset to train for each website of interest. He has the disadvantage, however, of not being able to compare precise timing information between two instances of the same connection. Further, WF classifiers have advanced substantially since the introduction of AP in 2006. We thus need to adapt AP to protect against WF in Tor.

In the rest of this section, we describe AP and explain how we adapt it to defend against WF attacks in Tor.

5.1 Design Overview

To clarify the notation adopted in this paper, we use outgoing to refer to the direction from the client to the web server, and conversely, incoming is the direction from the web server to the client.
The basic idea of AP is to match the gaps between data packets with a distribution of generic web traffic. If an unusually large gap is found in the current stream, AP adds padding in that gap to prevent long gaps from being a distinguishing feature. Shmatikov and Wang recognized the importance of bursts in web traffic and thus developed a dual-mode algorithm. In burst mode, the algorithm essentially assumes there is a burst of real data and consequently waits for a longer period before sending any padding. In gap mode, the algorithm assumes there is a gap between bursts and consequently aims to add a fake burst of padding with short delays between packets.

In the WF literature, a burst is typically defined as a sequence of consecutive packets in the same direction (incoming or outgoing). In this paper, we follow Shmatikov and Wang and define a burst in terms of bandwidth: a burst is a sequence of packets that has been sent in a short time period. Conversely, a gap is a sequence of packets that has been sent over a longer timespan. Then, the objective of AP can be described as adding padding to gaps such that bursts are not as prominent and distinguishable. We explain later in this section how to set the bandwidth threshold between bursts and gaps.

AP has multiple important effects in the WF scenario. First, we note that bursts of traffic have been shown to carry a great deal of page-identifying information [19, 9, 18], and they are key features in all recent WF attacks. AP adds new bursts of padding, thereby adding noise to burst patterns and making the page’s burst distribution closer to generic web traffic. It also adds padding randomly to some real bursts. This can occur when a fake burst generated in gap mode is interrupted by a real burst, which pads the beginning of the real burst (see Figure 4). Additionally, the padding changes important features such as the total page size, number of bursts, average burst length, and does so in an unpredictable way each time the page loads. This reduces the utility of training the classifier with these features.

**AP algorithm.** The AP algorithm is defined by two histograms of delays that we call $H_B$ (used in burst mode) and $H_G$ (used in gap mode). The histograms have a set of bins that spans over the range of possible inter-arrival times. Each bin contains a number of tokens, which can be interpreted as the probability of selecting an inter-arrival time within the range of delays represented by that bin. The last bin, which we dub the “infinity bin”, includes all possible values greater than the second-to-last bin. For more details on how these histograms are defined in WTF-PAD we refer the reader to Appendix A.

AP implements the state machine shown in Figure 5 in each defense endpoint, i.e. both the PT client and the PT server. For simplicity, let us consider just the client’s state machine, which covers traffic flowing to the PT server.

**Burst mode.** As depicted in the diagram, AP starts idle (state $S$) until transmission starts and it receives a real packet ($R$). This causes it to enter burst mode (state $B$), drawing a delay $t$ from the $H_B$ histogram. Then it starts to count down until either a real packet arrives or $t$ expires. In the first case, the real packet is immediately forwarded, a new delay is sampled and the process is repeated again, i.e. it remains in burst mode. Otherwise, a dummy message ($D$) is sent to the other end and AP switches to state $G$ (gap mode).

The $H_B$ histogram governs how AP reacts to bursts. It is built using a large dataset of web traffic, out of which we sample the times between the end of a burst and the beginning of the following burst (see Section 5.3). Therefore, while we are in a burst, the delays we sample from $H_B$ will not expire until we find an inter-arrival time that is longer than typical within a burst, which will make the delay expire and trigger the $G$ state.

**Gap mode.** While AP is in state $G$, it samples from histogram $H_G$ and sends dummy messages when the times it samples expire. The histogram for gap mode, $H_G$, is built from a sample of inter-arrival times within a burst in traffic collected for a large sample of sites. That is, by sending packets with inter-arrival times drawn from $H_G$, we are able to generate fake bursts that follow the timing distribution of an average burst.

A transition from $G$ back to $B$ occurs upon either sampling a token from the infinity bin or receiving a real packet. Similarly, a transition from $B$ to $S$ happens when we sample a token from the infinity bin.
Note that whenever AP receives a real packet, it immediately forwards it. Since sending a real packet means that the timeout expired, AP has to correct the distribution by returning the token to its bin and removing a token from the bin representing the actual delay. This prevents the combined distribution of padding and real traffic from skewing towards short values and allows AP to adapt to the current transmission rate [20]. If one bin runs out of tokens, to minimize any perturbation this could cause on the resulting distribution of real and dummy inter-arrival times, we remove tokens from the immediately non-empty greater bin [20]. In case all bins are empty, we refill the histogram with the initialization sample.

5.2 WTF-PAD

We propose a generalization of AP called Website Traffic Fingerprinting Protection with Adaptive Defense (WTF-PAD). WTF-PAD includes implementation techniques for use in Tor and a number of link-padding primitives that enable more sophisticated padding strategies than the basic AP described above. These features include:

Receive histograms. A key feature to make padding realistic is to send padding messages as a response to messages received from the other end. In WTF-PAD, we implement this by keeping another state machine like the one in Figure 5, but swapping incoming and outgoing traffic. For example, the client in this FSM has a rcv event when it gets a packet from the PT server (incoming). This allows us to encode dependencies between incoming and outgoing bursts and to simulate request-response HTTP transactions with the web server. Padding introduced by the rcv event further distorts features on bursts, as just one packet in the outgoing direction might split an incoming burst as considered by the attacks in the literature.

Control messages. WTF-PAD implements control messages to command the PT server padding from the PT client. Using control messages, the client can send the distribution of the histograms to be used by the PT server. This way, the PT client is in full control of the padding scheme. It can do accounting on received padding traffic and alert the user if relays in its circuits are sending unscheduled padding, which could mean they are exploiting padding to induce a side channel.

Beginning of transmission. Control messages can also be used to signal the beginning of the transmission. If we are in state $S$ and a new page is requested, we will need to flag the server to start padding. Otherwise, the transmission from the first request to the following response is uncovered and reveals the size of the index.html page.

Soft stopping condition. In contrast to Tamaraw and CS-BuFLO, WTF-PAD does not require an explicit mechanism to conceal the total time of the transmission. At the end of the transmission, the padding is interrupted when we hit the infinity bin in the gap state and then the infinity bin in the burst state. This means that the tokens in the infinity bins of $H_B$ and $H_G$ also govern the amount of padding appended to the end of the page, and the probability of stopping will depend on the shape of the histograms at the end of the transmission. See the Appendix A for further discussion on how to set the tokens in the infinity bins. The lack of a firm stop padding condition represents an advantage over existing link-padding-based defenses, which require a mechanism to flag the boundaries of the transmission. Given the prevalence of AJAX and dynamic content in the modern Web, TCP connections may remain open until the user closes the browser or the tab, and it is not trivial to decide when the session ends.

5.3 Inter-arrival time distributions

Shmatikov and Wang did not specify in the original AP paper how to build and use the distribution of inter-arrival times in the AP histograms. In their simulations, they sampled the inter-arrival times for both the real traffic and the padding from the same distribution. To build the histograms, we have sampled the times from a crawl of the top 35K pages in the Alexa list. First, we uniformly selected a sample of approximately 4,000 pages and studied the distribution of inter-arrival times within their traces. In order to implement WTF-PAD without revealing distinguishing features between real and fake messages, we need to send dummies in time intervals that follow the same distribution as real messages. In Figure 6, we observe that times for incoming and outgoing traffic have different distributions. The asymmetric bit rates in the connection we used to conduct the crawl account for this difference. Since WTF-PAD has different histograms in the client and the relay we can simulate traffic that follows different distributions depending on the direction.

Next, we explain how to find the bursts and the gaps in the inter-arrival time distribution and build the histograms $H_B$ and $H_G$. Intuitively, the burst-mode histogram $H_B$ should consist of larger delays covering the duration of typical bursts, while the gap-mode histogram $H_G$ should
consist of smaller delays that can be used to mimic a burst. To split inter-arrival times into the two histograms, we calculate the instantaneous bandwidth at the time of each inter-arrival time to determine if it is part of a burst or not. Then, we set a threshold on the bandwidth to draw the line between bursts and gaps.

We estimate the instantaneous bandwidth using a sliding window over a sequence of consecutive packets. We have experimented with different window lengths and threshold values. The best results against the state-of-the-art WF attack are achieved for a window of two consecutive packets and a threshold set to the total average bandwidth for the whole sample of traces.

5.4 Tuning mechanism

AP can hide inter-arrival times that are longer than the average, but it does not hide times that are shorter than the average. To effectively hide these times we need to either add delays to the problematic traces or add more padding over all traces to level off these times and make them less distinctive. We focus on the latter because our objective is to minimize delay. WTF-PAD provides a mechanism to tune the trade-off between bandwidth overhead and security: one can modify the parameters of the distributions used to build the histograms to add more padding and react to shorter inter-arrival times.

![Figure 7: Histogram of times between consecutive bursts](image)

Figure 7: Histogram of times between consecutive bursts for incoming traffic. In dark gray we superpose the pdf of our log-normal fit. In light gray, we show the pdf of a shifted log-normal distribution that we use to build the $H_B$ histogram.

To illustrate this, we show in Figure 7 the bins of the $H_B$ histogram as sampled from our dataset. We observe that the distribution of the logarithm of these times can be approximated with a normal distribution $\mathcal{N}(\mu, \sigma^2)$. That is, the inter-arrival times follow a log-normal distribution. We can modify its mean and variance to obtain a new normal distribution $\mathcal{N}(\mu', \sigma'^2)$ that we will use to sample the inter-arrival times of $H_B$. We can do this transformation in the log domain as a normal distribution because the associated log-normal distribution will have the same parameters.

In the figure, the bins on the right that stand above the curve of the probability density function (pdf) of our fitted distributions contain times that are likely to be observed in an unprotected transmission. However, the times sampled by AP from $\mathcal{N}(\mu, \sigma^2)$ are likely to be shorter, and thus will replace the expected inter-arrival times in the distribution observed by the attacker. By using $\mathcal{N}(\mu', \sigma'^2)$ we are shifting the average distribution of inter-arrival times toward even shorter values. This results in a greater amount of short times being covered by padding, which increases the bandwidth overhead but causes the pages become less distinguishable and thereby reduces the attacker’s accuracy.

We created a statistical model of the underlying distributions of inter-arrival times from the samples we extracted from our dataset. We experimented with multiple positively skewed distributions such as Pareto, Weibull, Gamma, and Beta to build the model and test the goodness of fit with the Kolmogorov-Smirnov. We estimated the parameters of the distributions using maximum likelihood estimation. Even though Pareto and Beta distributions seemed to fit best, we decided for simplicity to use normal and log-normal distributions, given that the error was not significantly greater than that observed in the other distributions.

To calibrate the possible shifts, we set $\mu'$ and $\sigma'$ according to the percentile of the real data we want to include. For instance, assuming a normal distribution, if we adjust $\mu'$ to the 50th percentile, we obtain $\mu' = \mu$ and $\sigma' = \sigma$. If we set $\mu'$ to the value of the pdf at the 10th percentile, we then derive the $\sigma'$ using the formula of the pdf of the normal distribution.

This tuning mechanism is applied only on $H_B$, as it defines the probability of injecting padding inside bursts as well as starting a fake burst after a real burst ($H_C$ defines the shape of fake bursts).

6 Evaluation

In this section we discuss how we evaluated WTF-PAD, present our findings and compare them with the results we obtained for existing defenses.

6.1 Data

Unlike most previous defense evaluations, which used simulated data, we have used web traffic that has been collected over Tor. We used a dataset that had been collected for a study about a realistic evaluation of WF attacks [14].
Table 1: Performance and security comparison among link-padding defenses (closed-world).

| Defense         | Parameters                        | Accuracy (%) | Overhead (%) |
|-----------------|-----------------------------------|--------------|--------------|
| BuFLO [9]       | $\tau = 10s, \rho = 20ms, d = 1500B$ | 14.9         | 14.1         |
| CS-BuFLO [2]    | $\rho = [20, 200]ms, d = 1500B, CPSF$ | 30.6         | 40.5         |
| Tamaraw [25]    | $\rho_{out} = 0.053, \rho_{in} = 0.138, d = 1500B$ | 13.6         | 10.59        |
| WTF-PAD         | Normal fit, $p = 0.4, d = 1500B$   | 17.25        | 15.33        |

In their work, the authors studied several variables in isolation, including website variance over time, multi-tab browsing behavior, Tor Browser Bundle (TBB) version and Internet connection. The authors collected multiple sets of data for different values of these variables. They observed that when assumptions on these variables are violated, the accuracy of existing attacks drops significantly. Our purpose in the present work is to evaluate defenses, instead of attacks, under similar conditions.

The list of URLs used for crawling was the Alexa 35,000 most popular websites [1]. The datasets were crawled in ten batches and in each batch all the pages were visited four times. A traffic trace of a visit to a page is modeled as a sequence of inter-arrival times and (Ethernet) packet lengths. The inter-arrival times are expressed with floating point representation and the packet lengths are integers, where the sign of each integer represents the direction of the packet: negative for incoming and positive for outgoing.

6.2 Methodology

To evaluate the improvements in performance offered by the defense, we applied the attack’s classifier on both the original traffic traces and traces that have been protected by applying the defense. The classifier is first trained and tested with the non-protected traces and then trained and tested with the protected ones. We compare both results to get an estimation of the level of protection the defense gives against the attack.

We used the framework described in Section 4 to simulate WTF-PAD on the traces of our dataset. The difference in bandwidth between the original trace and the protected trace provides us with an estimate of the overhead. Analogously, the difference in time gives us an estimate for the latency overhead. Given the probabilistic nature of AP, we run each experiment several times to cope with the variance of the estimators. We measure the median, the mean and the standard deviation of the results over multiple repetitions of the experiment.

We applied the state-of-the-art attack on the set of protected traces to evaluate the effectiveness of the defense. The accuracy of the attack determines the security provided by the defense. In the closed world, we measure the accuracy as the True Positive Rate (TPR), or Recall. We also measure the False Positive Rate (FPR), the Positive Predictive Value (PPV)—also called Precision, and the harmonic mean of precision and recall (F1-Score), as they play an important role on evaluating the effectiveness of the attack in the open-world setting. We also used the ROC (Receiver Operating Characteristic) and the Precision-Recall curves, in the close and open world respectively, to evaluate the performance of the classifier and the impact that the defense has on it. Later in this section we explain why these metrics are more appropriate than other metrics used in prior work.

The state-of-the-art attack is based on a k-NN model [25]. k-NN is a supervised learning algorithm, meaning that it constructs hypotheses on data that have been labeled beforehand. When we feed the classifier a test instance, it measures the distance from that instance to all training instances. Next, it selects the $k$ closest instances (the neighbors, and outputs the class of the majority of the neighbors as its guess. Wang et al. determined by cross-validation that the number of neighbors that optimizes the trade-off between true positives and false positives is $k = 5$. The distance defined by Wang et al. for use in k-NN is a weighted sum of a set of features. This feature set is the most extensive in the WF literature including more than 4,000 features. Some of the features are constructed based on variations of a parameter that defines a family of features. An example is the number of bursts with length longer than $N$ packets, for $N = 5, 10, 15$.

In the literature, we see that attacks play a game of cat and mouse against defenses by either exploiting features that existing defenses do not protect or by looking for new features [4]. On the other side, in order to save bandwidth, defenders conceal only those features that attackers try to exploit. The weights in the k-NN distance allow for a tuning of the metric that gives more relevance to those features that contain more identifying information. This way, the learning model is robust to perturbations introduced by the defenses on only a subset of features [25].

Another reason for which we chose the k-NN based attack is that its features extensively exploit bursts in traffic. We observed that the weight learning function of the attack assigns zero weight to all the features that are not re-
lated to bursts, implying the defense effectively conceals these features. The weights for burst features decrease but are still greater than zero. In Appendix B, we show the results for our analysis of the impact of WTF-PAD on these features using the k-NN weight learning function.

In order to have a comprehensive evaluation of WTF-PAD, we also evaluated it with other existing WF attacks that take into account features that are not included in the set of features of k-NN (see Table 1).

6.3 Results

To evaluate the trade-off between bandwidth overhead and accuracy provided by WTF-PAD, we applied the attack on instances protected with it tuned using different percentile values, ranging from 0.5 (low protection) to 0.01 (high protection) percentiles.

In Figure 8, we show the trade-off curves for both normal and log-normal fits. We observe a steeper decrease in accuracy for the normal model with respect to the log-normal one. Remarkably, beyond a certain point (around 0.1 percentile), the tuning mechanism saturates to 15% accuracy for both models: percentiles lower than that point do not further reduce accuracy and only increase bandwidth overhead. The trend we observe is the cost in bandwidth exponentially growing with the protection level that the defense attempts to provide.

As we see, WTF-PAD is the only defense to provide zero latency overhead. The other defenses we tested produce between 145-200% additional average delay to fetch a webpage. WTF-PAD also offers moderate bandwidth overhead. For our datasets, we observed that the bandwidth overhead was always below 60% while attaining decreases in the accuracy of the attack that are comparable with the other defenses.

ROC curve. To study the impact of WTF-PAD on the performance of k-NN, we also plotted the ROC curve with and without protection. The ROC curve represents the performance of the classifier when its discrimination parameter changes. The standard k-NN is not a parametric algorithm, meaning that there is no explicit parameter that one can use to set the threshold and tune the trade-off.

We have defined more or less restrictive classifications of k-NN by setting a minimum number of votes required to classify a page as monitored. For example, at one extreme, we may require that all neighbors must vote for the monitored class to label it as monitored, thus reducing false positives but also reducing the true positives (detection rate). At the opposite extreme, we may require only one vote to assign the monitored label, increasing both the detection rate and the false positive rate.

The way ROC diagrams are interpreted is as follows. A classifier that outputs a class uniformly at random has a curve corresponding with the diagonal TPR=FPR. Any classifier that is above this ‘random’ curve performs better than random guessing and the best classifier is close to the top-left corner of the graph. The area under the ROC curve summarizes the performance of the classifier and can be used to compare classifiers that are evaluated on the same problem.

We used 10-fold cross-validation to average the ROC curve for $k=5$ neighbors in a closed world of 100 pages. To plot the ROC of a classification one has to translate the problem into a binary problem. For that end, we divided the set of pages into two halves, 50 monitored and 50 non-monitored, and considered the monitored as the positive class and the non-monitored as the negative one. Then, all the positive (monitored) observations that are classified as a page in the positive class are counted as true positives, even if the instances were classified as a different monitored page. This is a more advantageous scenario for a surveillance-type of attacker that only tries to identify whether the page is monitored or not. In practice, we observed in our experiments that the rate of pages that are classified as a different page in the monitored class is one-to-two orders of magnitude lower than the TPR for multi-class classification.

It is important to note that in this binary classification the k-NN still works as a multi-class classifier, but the way we count TP, FP, FN and TN is different. In Appendix C we show a table with the confusion matrices used in the classification problems of this study.

The curves in Figure 9 are the mean ROC curves for the data before and after applying the defense with respect to random guessing. To ensure a balance between the positive and negative classes, in each fold of the 10-fold cross-validation we trained and tested using the same
number of instances for monitored and non-monitored. We notice a significant reduction in the performance of the classifier with the set of protected traces. Compared to unprotected data with an AUC of 0.95 (close to perfect classification), WTF-PAD has an AUC of 0.66, which is substantially closer to random guessing.

7 Realistic Scenarios

In this section, we present the results of the evaluation of the defense in two realistic scenarios: the open world and the use of multi-tab browsing.

7.1 Open-world evaluation

We now evaluate the performance of the defense against the k-NN algorithm in the open-world scenario. Our definition of the open-world is similar to the ones described in prior work. Under this definition, the goal of the attacker is to find out whether the page the user is visiting falls in a list of monitored pages or not. We have evaluated the k-NN with the evaluation method used by Wang et al. and incorporating the changes suggested by Wang [24], so that we can compare our results with the ones they obtained [25]. We also have developed a new evaluation which we believe complements the results presented by Wang et al.

Wang’s dataset contains 90 instances for each of 100 monitored pages and only one single instance for each of 9,000 non-monitored pages [24]. Wang et al. measured the accuracy of the k-NN classifier in detecting monitored instances that it has not trained on using leave-one-out cross-validation. In leave-one-out, only one instance of the data is held out for testing, and the rest of the instances are used for training, including instances of the open-world.

In their open-world classification, they consider one class for each of the monitored pages and one single class for all the non-monitored pages. Then, the attacker aims to identify the exact monitored pages the user visits and to classify all the visits to non-monitored pages into the non-monitored class regardless of the actual page. We reproduced their experiments with our dataset of 40 instances per monitored page and a single instance for each of 15,000 non-monitored pages.

Figure 10 plots several classification performance metrics as a function of the world size used to train and test the classifier. We observe that even though the accuracy initially increases as the world grows and saturates to 90% at the maximum considered world size, the F1-Score decreases and levels off to 50%. This is because even though the FPR rapidly drops to zero, the TPR decreases below 40%. The accuracy is over 90% because the classifier reaches almost perfect classification for the non-monitored class.

This high accuracy is due to the stringent threshold used in the k-NN which requires all neighbors to vote to the same class. The threshold effectively reduces the amount of false positives but the imbalance of the open-world dataset, where the monitored instances are a small fraction of the total, increases the TNR to its maximum. Basically, because of this imbalance, a classifier that outputs a class uniformly at random would already obtain high accuracy detecting negative instances.

We observe that the TPR and FPR after applying the defense are dramatically lower than the rates shown in Figure 10. However, due to the skew between the positive and the negative classes, the ROC curves of the k-NN are biased towards the negative class and do not reflect well the performance of the classifier. For imbalanced datasets,
it is recommended to use the Precision-Recall ROC (P-ROC) instead of the ROC [7]. Similarly to the standard ROC, P-ROC represents the interaction of TPR (recall) and PPV (precision), instead of FPR, with respect to variations on the discriminant of the classifier. Precision is the number of true positives divided by the number of instances classified as positive, a metric that in the open-world scenario conveys the fraction of monitored pages that were correctly detected by the k-NN. Precision is invariant to the size of the negative class and thus gives a more accurate estimation of the classifier's performance in the open-world.

The P-ROC has a different interpretation to the ROC curve. In this case, the perfect classifier has a curve that coincides with the top-right corner and the random classifier is calculated as the number of positives divided by the total number of instances, i.e., the probability of selecting a positive instance uniformly at random. This random curve is used as a baseline because no classifier can have lower precision than it and, like in the standard ROC, classifiers can be benchmarked by comparing their area under their curves (AUC).

Figure 11 shows the P-ROC curve of the k-NN when applied on the set of traces before and after WTF-PAD protection. We used the same threshold method as for the ROC curve and used leave-one-out to follow Wang’s methodology. Again, we observe that the AUC for the unprotected case is reduced significantly (from 0.79 to 0.27) and is close to random.

Figure 11 is a snapshot of the performance of the classifier for a world size 5,000. It shows that the performance of the k-NN attack is reduced by the defense, but it is not informative about how the size of the world affects the k-NN for the unprotected and protected data. In Figure 12 we plot the AUC estimates for both cases by varying the size of the open world. The first data point represents a closed world where all pages are monitored and, naturally, we see that all classifiers perform as in perfect classification (AUC=1).

However, as we increase the size of the world, the baseline classification tends to zero because monitored pages become less likely to occur and thus a random guess is less likely to succeed. On the other hand, the k-NN levels off to AUC 0.74, which means that it is not heavily affected by the size of the world. Notably, when we apply the defense on the traces, all AUC values are close to random even for the largest world size that we have considered (15K pages). Even though the k-NN tends to stabilize to a constant with the size of the open world, WTF-PAD steadily decreases its performance at the same rate as the random classifier does.

Note that by selecting all pages uniformly from the open-world we are simplifying the model of the Web. As other studies have pointed out, a more accurate model could be built if we had more information about the browsing habits of Tor users [14]. In our model we consider the case in which monitored pages are less likely to occur than non-monitored pages, which is realistic in cases in which a surveillance adversary tries to monitor pages that are not popular, as they would reveal more personal information than the ones that are visited by a large fraction of the population.

### 7.2 Multi-tab evaluation

In this section, we assume the victim is using Tor for web browsing, with no other applications running over Tor, but she opens multiple tabs and windows that will
establish multiple concurrent HTTP sessions. Opening multiple tabs is a common practice, and even though a tab may seem to be finished loading, it still might generate traffic by AJAX requests and other dynamic content.

The objective of the experiments in this section is to evaluate the efficacy of the WTF-PAD defense when the user is browsing with multiple tabs open. For this evaluation, we considered two scenarios. For Scenario 1, following the method of Juarez et al. [14], we consider an attacker that uses single tab data to train his classifier but, instead of testing on just multi-tab traces, we tested on a combination of single-tab and multi-tab traces with different time offsets between tabs. This test set is reasonable because, even though the attacker will not be able to train on all possible combinations of pages in the multi-tab, there will be a certain amount of traffic that is single tab. For Scenario 2, we assumed the attacker can detect the number of tabs that are open and will thus train on multi-tab instances for that number of tabs. In both scenarios, the goal of the attacker is to identify one of the pages that compose the traffic trace.

Table 2: TPR for protected and unprotected traces in Scenarios 1 and 2.

|        | Unprotected | WTF-PAD |
|--------|-------------|---------|
| Scenario 1 | 14%         | 8%      |
| Scenario 2 | 68%         | 22%     |

To evaluate Scenario 1, we trained the k-NN attack on a single-tab dataset and tested on a mixed dataset of single tab traces and multi-tab traces generated by a crawl with two simultaneous tabs. The first tab was loaded following the Alexa top 100 sequentially. The second tab was open with a delay from 0.5 to 5 seconds and was chosen uniformly at random from the same list4. In the evaluation of this scenario, we used 10-fold cross-validation to suppress the effect of overfitting. Table 2 shows the result of Scenario 1 for traces both with and without the protection offered by WTF-PAD.

Since the accuracy of the k-NN is already low when training on single-tab and testing on multi-tab (Scenario 1 in Table 2), the defense does not impact significantly the TPR of the classifier. As shown in Table 3, the share of single-tab traces in the test set is small compared to the total number of multi-tab traces. If we consider single-tab traces without protection, k-NN successfully detects them with 78% accuracy, which is roughly the TPR that k-NN achieves in the closed-world scenario. The accuracy in detecting single tab traces protected by WTF-PAD is 31%.

To evaluate Scenario 2, we trained and tested k-NN on a dataset that includes multi-tab and single-tab traces.

Table 3: TPR with respect to each traffic type. Column First shows the number of background pages (the first tab) detected among truly detected multi-tab traces.

|                  | Scenario 1 (TP/Total) | Scenario 2 (TP/Total) |
|------------------|-----------------------|-----------------------|
|                  | Single | Multi | First | Single | Multi | First |
| unprotected      | 233/300 | 901/8100 | 544/901 | 263/300 | 482/810 | 449/482 |
| WTF-PAD          | 95/300 | 598/8100 | 333/598 | 108/300 | 137/810 | 103/137 |

The same as Scenario 1, we used 10-fold cross validation to evaluate Scenario 2.

Table 2 shows the accuracy of the k-NN classifier in Scenario 2. In this scenario, the attack achieves 68% TPR on unprotected multi-tab traces, much higher than the 14% found in Scenario 1. However, the success rate on protected traces drops to 22%. Table 3 shows a breakdown of the detection rates over the types of traffic used to build the test set. k-NN could successfully classify unprotected single-tab traces with an accuracy of 87%, which is close to the accuracy rate of k-NN in the closed-world setting. The accuracy decreases to just 36% when we protect the traces with WTF-PAD.

In both multi-tab scenarios, we found that k-NN could detect the background page (the first tab) with much greater accuracy than the foreground page (the second tab). Additionally, adding significant delay before loading the foreground page increases the accuracy of classifying it. The likely cause of these findings is that the first few packets in a page often hold important features. These features can be used to best identify the background page, which is initially loaded with no interference from the foreground page, while the first few packets of the foreground page are typically masked by traffic from the background page that is still loading.

8 Discussion

Website fingerprinting is a significant threat to the security of Tor users. WF attacks fall within the Tor threat model [8], as it only requires one point of observation between the client and the bridge or guard, and the attack potentially deanonymizes users by linking them with their browsing activity. Even with the challenges of open-world and multi-tab browsing [14], some websites may exhibit especially unique traffic patterns and be prone to high-confidence attacks. Attacks may observe visits to the same site over multiple sessions and to gain confidence in a result. Further, an attacker may use WF to confirm a-priori information, which is problematic even if it does not fall under the Tor attack model (which specifically excludes confirmation attacks).

Protecting Tor users from WF attacks, however, must be done while maintaining the usability of Tor and lim-
iting costs to Tor relay operators. Delay is already an issue in Tor, so adding additional delay would harm usability significantly. The BuFLO family of defenses add between 145-200% additional delay to the average website download, i.e. up to three times as long to get a webpage, which makes them very unlikely to be adopted in Tor.

The main overhead in WTF-PAD is bandwidth, which was under 60% overhead in all scenarios we tested. We do not know the exact percentage that is acceptable for use in Tor, but we note the following points. First, approximately 40% of Tor traffic is bulk downloads (from 2008, the last data we know of) [17]. To the extent that this holds today, only the remaining 60% of traffic needs to be covered by this defense. Second, the bottleneck in Tor bandwidth today is exit nodes. WF defenses do not need to extend to exit nodes, stopping at the bridge (in our framework) or at the guard or middle node when fully implemented [5]. Thus, the bandwidth overhead only extends to one or two relays in a circuit and crucially not to the most loaded relay, making the overhead cost much less in practice. Third, given our findings for the open-world setting, it may be possible to tune WTF-PAD further to lower the bandwidth and maintain useful security gains in realistic use cases.

8.1 Open Issues in Realistic Scenarios

Even though we have studied the open-world and multi-tab scenarios, we lack the necessary information to reproduce these scenarios accurately. First, although we can guess that Tor users are likely to use multi-tab browsing due to additional delays and the prevalence of multi-tab use on browsers like Firefox, we do not know the actual patterns of multi-tab browsing for Tor users. Second, we do not know which sites Tor users are going to and with what frequencies, which could be different from Alexa top sites. This matters significantly to the open-world scenario, for which the most frequently visited sites would be the most likely to be a false positive, but only if they appear similar to targeted sites.

It may be possible to carefully collect such data from exit nodes or conduct explicit studies of Tor users. Both of these approaches may have significant drawbacks, unfortunately, and remain a part of future work.

8.2 Limitations of WTF-PAD

The construction of the histograms \( H_B \) and \( H_C \) is critical for the correct performance of AP. Our method for building these histograms have the following two main limitations.

First, our statistical models have been built using basic distributions (e.g., normal, log-normal) that overfit the underlying distribution of inter-arrival times to our dataset. Researchers in network performance optimization have obtained good results in modeling traffic burstiness using a model called *Poisson Pareto Burst Process* [28]. We think that developing more sophisticated models of network inter-arrival times can improve the performance of WTF-PAD.

Second, since the distribution of inter-arrival times depends on the connection of the client, we cannot estimate them a priori and ship them with WTF-PAD. A solution to this problem is to consider groups of clients with similar connections and have a precomputed configuration for each group. Then, the clients will estimate the properties of their network during a bootstrap phase and only download the configuration that best matches their connection.

We believe that further research on evaluating link-padding based schemes against WF should attempt to draw the line between what is realistic and what is not. The threat model of WTF-PAD should be extended to protect against malicious entry points, either guards and bridges. The evaluations should be done by implementing WTF-PAD in Tor and running it on a set of test Tor nodes.

9 Conclusion

In this paper, we described the design of WTF-PAD, a probabilistic link-padding defense based on Adaptive Padding. We studied the effectiveness and overheads of WTF-PAD, and compared it to existing link-padding-based defenses, showing that it offers reasonable protection with lower overhead costs. In particular, our results show that WTF-PAD does not introduce any delay in the communication while introducing moderate bandwidth overheads, which makes it especially suitable for low-latency communications such as Tor. Additionally, we have evaluated the effectiveness of WTF-AP in open-world and multi-tab scenarios. The results show that the defense reduces the performance of the classifier to random guessing.

For this evaluation, we have developed a framework that allows researchers to evaluate the traffic analysis resistance of Pluggable Transports. With this framework, future researchers on traffic analysis in Tor can crawl the web, and simulate their traffic analysis resistance protocols as implemented in a Tor Pluggable Transport.

References

[1] Alexa. Alexa Top 500 Global Site. http://www.alexa.com/topsites, 2015.
[2] Cai, X., Nithyanand, R., and Johnson, R. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In Workshop on Privacy in the Electronic Society (WPES) (2014), ACM, pp. 121–130.
A histogram is defined as a disjoint partition of the support of a random variable. To construct a histogram, the range of the variable is divided into a finite number of intervals or "bins." The frequency of the variable in each bin is then counted. The resulting information is often displayed in a graphical format called a histogram.

In the context of side-channel analysis, histograms are used to analyze the distribution of side-channel leakage. For example, the frequency of the leakage in different bins can be used to infer information about the underlying computation. This is because the leakage is often correlated with the operation being performed, and by analyzing the distribution of the leakage, it is possible to gain insights into the operations.

Thus, histograms provide a powerful tool for analyzing side-channel leakage and can be used to develop defenses against such attacks. However, it is important to note that the effectiveness of these defenses depends on the specific characteristics of the leakage and the side-channel attack.

Notes

1. If the defense was implemented in the Onion Router, one could use the PADDING and VPADDING cells that exist in the specification of the Tor protocol: https://gitweb.torproject.org/tor-spec.git/tree/tor-spec.txt.

2. It has been shown that constant-rate padding may reveal information about the CPU load and the client’s real transmission rate [10].

3. The increase in bandwidth use could add moderate delays in practice, particularly for clients which bandwidth are surpassed by the bandwidth overhead introduced by WTF-PAD.

4. We note that the base rate of single versus multi-tab is critical to evaluate the performance of the classifier but, since we do not have statistics about the distribution of these different types of traffic, we have used arbitrary proportions of testing and multi-tab in the test set.

5. There already exists a proposal to modify Tor’s load balancing equations to take into account padding: https://gitweb.torproject.org/tor-spec.git/tree/proposals/265-load-balancing-with-overhead.txt.

A WTF-PAD Histograms

A histogram is defined as a disjoint partition of the support of the inter-arrival time distribution \([0, +\infty)\). Each sub-interval, that we call \(bin\), is a half-closed interval \(I_i = [a_i, b_i)\) with \(0 \leq a_i, b_i < +\infty\) for all \(i = 1, \ldots, n\), where \(n \geq 2\) is the total number of bins in the partition. The bin lengths used in the AP histogram increase exponentially with the bin index, namely, the intermediate bins have the following endpoints:
\[ a_i = \frac{M}{2^{n-i}}, b_i = \frac{M}{2^{n-i-1}}, \]

for \( i = 2, \ldots, n - 1 \). \( M > 0 \) is the maximum inter-arrival time considered in practice. The first bin is \( I_1 = [0, \frac{M}{2^n}) \) and the last bin is \( I_n = [M, +\infty) \).

An exponential scale for the bins provides more resolution for values in a neighborhood of zero, which is convenient to represent distributions with heavy positive skew, such as the distribution of inter-arrival times in network traffic.

When we sample from a bin, AP returns a value sampled uniformly from \([a_i, b_i]\), except for the last bin \([M, +\infty)\), in which case AP returns “\( \infty \)”.

In Figure 13, we show a simplified version of the histograms we used in the WTF-PAD instance at the client. The histograms that we actually used have 20 bins.

Each bin contains a number of tokens \( k_i \). We denote \( K \) the sum of tokens in all the bins except the infinity bin, i.e.:

\[ K := \sum_{i=1}^{n-1} k_i. \]

If we assume the probability of selecting a token is uniform over the total number of tokens, then the probability of sampling a delay from that bin can be estimated as:

\[ P_b := \frac{k_i}{K + k_n}. \tag{1} \]

We assume that all the bins \( I_i \) for \( i < n \) are already filled. In the following we describe how to set the number of tokens in \( I_n \), the infinity bin, for both histograms, \( H_B \) and \( H_G \).

**Infinity bin in \( H_B \).** According to the notation introduced above, \( P_b \) in \( H_B \) is the probability of falling into the infinity bin and thus defines the probability of not sending padding (and not starting a fake burst) when we draw a sample from it. To express \( k_n \) in terms of the probability of sampling from \( I_n \) and the current sum of tokens in the histogram, we clear the expression of \( P_b \) in Equation 1 for \( k_n \):

\[ k_n = \frac{P_n}{1 - P_b} K. \]

For instance, if we decide on setting the probability of generating a fake burst to 0.9, then we need to set \( P_n = 0.1 \). Assuming \( K = 300 \) tokens, using the equation above we obtain \( k_n \approx 34 \).

**Infinity bin in \( H_G \).** The number of tokens we will sample from \( H_G \) until we hit the infinity bin is the number of dummy messages we will send within a fake burst. Since the probability of drawing a token is uniform, we can think the histogram as one single bucket that contains tokens from \( I_n \) and tokens from the other bins. Then, the expected number of draws without replacement, \( L \), until we draw the first token from the infinity bin is a known result one can find in any probability textbook:

\[ E[L] = \frac{K + k_n + 1}{k_n + 1}. \]

We know the expected value of the length of a burst from our estimations on a large dataset of web traffic. Let \( \mu_L \) be the mean burst length. In order to make sure fake bursts have the same mean length as real bursts, we must impose the expected number of tokens we sample until we hit the infinity bin to be:

\[ E[L] = \mu_L. \]

Then, we only need to clear \( k_n \) from the equation:

\[ k_n = \frac{K - \mu_L + 1}{\mu_L - 1}. \]

We will always set a large number of tokens compared to the length of a burst: \( K >> \mu_L \).

![Figure 13: Example of WTF-PAD histograms at the client. The histogram on the top is the \( H_B \) and the one at the bottom is \( H_G \).](image-url)

**B Feature analysis**

The k-NN attack proposed by Wang et al. uses a weight adjustment algorithm that reduces the weights in the distance used by k-NN for those features that are less informative [25]. The weight of a feature conveys how relevant that feature is for classification. We used this algorithm to study to what extent these features are concealed by our defense. To see the effect of WTF-PAD on the feature weights, we extracted the weights by applying the weight learning algorithm on both protected traces and unprotected traces.

The k-NN feature set consists of approximately 4,000 features, ranging from general features like total transmission time to features related to bursts in traffic. Around 3,000 of these features are related to the size of incoming and outgoing packets. We removed all the features related to the packets sizes from the feature set because these features were completely cancelled by the defense, as WTF-PAD pads all packets to Tor’s cell sizes.
As shown in Table 4 the weights drop dramatically after protecting the traces with WTF-PAD. The number of outgoing packets within each 30 packets and the first 300 outgoing packets have the highest weights on the protected traces. WTF-PAD is able to conceal most of the burst-related features such as the number of bursts in the traffic has a high weight on unprotected traces. Total transmission time is among the highest weight features in unprotected traces. In order to see the effect of the total transmission time on the classification, we applied the k-NN to the unprotected traces with only total transmission time as a feature and the detection rate of the k-NN reached 4%. This means that this feature is not distinctive of the page in our dataset. For this reason, we did not use the soft stop condition of WTF-PAD in our experiments.

Table 4: Weights associated with each feature during weight learning in kNN

| Features                                     | Weights |
|----------------------------------------------|---------|
| Outgoing packets within every 30 packets     | 104.7   |
| Outgoing packets between two outgoing packets| 159.2   |
| Total transmission time                      | 101.1   |
| The number of bursts greater than 15         | 31.9    |
| The number of bursts greater than 10         | 42.4    |
| 5th burst                                    | 32.2    |
| 4th burst                                    | 175.4   |
| Maximum burst                                | 18.5    |
| The number of bursts greater than 5          | 52.8    |
| The average number of bursts                 | 104.7   |
| The 3rd burst                                | 61.8    |
| The number of bursts                         | 107.5   |
| Total number of outgoing packets             | 92.2    |
| Total number of packets (trace size)         | 21.3    |
| Total number of incoming packets             | 24.3    |
| 2nd burst                                    | 0.1     |
| 1st                                          | 0.0     |

| Features                                     | Unprotected | Protected |
|----------------------------------------------|-------------|-----------|
| Outgoing packets within every 30 packets     | 0.03484     |
| Outgoing packets between two outgoing packets| 0.02952     |
| Total transmission time                      | 0.01705     |
| The number of bursts greater than 15         | 0.01092     |
| The number of bursts greater than 10         | 0.00734     |
| 5th burst                                    | 0.00535     |
| 4th burst                                    | 0.00471     |
| Maximum burst                                | 0.00419     |
| The number of bursts greater than 5          | 0.00290     |
| The average number of bursts                 | 0.00132     |
| The 3rd burst                                | 0.00124     |
| The number of bursts                         | 0.00088     |
| Total number of outgoing packets             | 0.00073     |
| Total number of packets (trace size)         | 0.0006      |
| Total number of incoming packets             | 0.00058     |
| 2nd burst                                    | 0.0         |
| 1st                                          | 0.0         |

Table 5: Confusion matrix in binary classification.

| Predicted page | True page $X$ ∈ monitored | $Y$ ∈ monitored | $Y$ ∈ non-monitored |
|----------------|----------------------------|-----------------|---------------------|
| True page $X$ ∈ monitored | TP                        | FN              | FN                  |
| True page $X$ ∈ non-monitored | FP                        | FP              | TN                  |

Table 6: Confusion matrix in multi-class classification.

| Predicted page | True page $X$ ∈ monitored | $Y = X$ | $Y$ ∈ non-monitored |
|----------------|---------------------------|--------|---------------------|
| True page $X$ ∈ monitored | TP                       | FN     | FN                  |
| True page $X$ ∈ non-monitored | FP                       | FP     | TN                  |

C Confusion matrices

In this study we have considered two different types of classification. Each type of classification corresponds to a different definition of the WF problem. In this section, we provide the definition of true positives, false positives, false negatives and true negatives for both of them.

**Binary classification.** We “binarized” the classification whenever we had to evaluate the performance of the attack using an ROC diagram. In this binary classification, we assumed the attacker trains the classifier on set of monitored and non-monitored pages, and tests it on the monitored pages and non-monitored pages which the classifier is not trained on. The attacker is interested in knowing whether the user is visiting one of these monitored pages or not, but he does not attempt to detect the exact page. Table 5 shows the confusion matrix for this classification. In this table, $X$ indicates the visited page and $Y$ the predicted page. We denote the set of monitored page as monitored and the set of non-monitored page as non-monitored. As shown in Table 5, a true positive happens when the true label and the predicted label are in the monitored set.

**Multi-class classification.** For this classification, we followed Wang’s strategy in [25] and used multi-class classification. Like binary classification, we assume the attacker trains the classifier on both monitored pages and non-monitored pages, and tests it on the monitored pages and a set of non-monitored pages which he is not trained on. In this scenario the attacker is interested in detecting the exact monitored page visited by the user, and is not concerned about identifying the actual non-monitored pages. Table 6 shows the confusion matrix for the multi-class classification. A true positive here is achieved when the true label and predicted label are exactly the same.