WTF-PAD: Toward an Efficient Website Fingerprinting Defense for Tor

Marc Juarez[1], Mohsen Imani[2], Mike Perry[3], Claudia Diaz[1], Matthew Wright[2]

[1]KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium
{marc.juarez,claudia.diaz}@esat.kuleuven.be

[2]The University of Texas at Arlington, TX, USA
mwright@cse.uta.edu, mohsen.imani@mavs.uta.edu

[3]The Tor Project, https://torproject.org
mikeperry@torproject.org

***Abstract*—*Website Fingerprinting* attacks are conducted by a low-resource local adversary who observes the (encrypted) traffic between an anonymity system and one or more of its clients. With this attack, the adversary can 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 effective, but they are impractical for deployment in real-world systems due to their high cost in terms of both added delay to access webpages 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 show that there are lightweight defenses that provide a sufficient level of security against website fingerprinting, particularly in realistic evaluation settings. In particular, we propose a novel defense based on *Adaptive Padding*, a link-padding strategy for low-latency anonymity systems that we have adapted for Tor. With this countermeasure in a closed-world setting, we observe a reduction in 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 with 50 pages that the attacker is monitoring and 2,000 other websites that users visit, 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's *Pluggable Transports* that we hope will contribute to future research on Tor and traffic analysis.

I. Introduction

Over the last decade, *Tor* has established itself as the largest deployed system for anonymously browsing the Web. As of today, it comprises a network of over six thousand relays and provides service to more than two million daily users [1]. Tor is designed to robustly foil local adversaries that aim to find out which webpages users browse. In particular, it uses a layered-encryption technique called *onion routing*, so that neither individual routers nor localised network eavesdroppers can link users to the pages they are visiting [2].

Tor is known to be vulnerable to a traffic analysis attack known as *Website Fingerprinting* (WF). WF attacks have received increasing attention in the last five years, driven by the high success rate reported in a series of recent studies [3], [4], [5], [6], [7]. State-of-the-art WF attacks achieve more than 90% accuracy against Tor [5], [6], [7] and require only a local eavesdropper or compromised guard node, thus breaking the anonymity properties that Tor aims to provide to its users.

A broad range of defenses has also been proposed to counter these attacks [4], [5], [7], [8], [9], [10], [11], [12], [13], [14]. The key building block of most of these defenses is *link padding*. Link padding adds varying amounts of latency 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 bandwidth and latency overheads.

State-of-the-art link-padding defenses impose bandwidth overheads between 130% and 400%, and increase latency so that pages load on average between two and four times slower. In exchange, they reduce the accuracy of the attacks, though not to the level of random guessing [8], [13]. These results are discouraging with respect to finding defenses that provide adequate protection at an affordable cost for a deployed system such as Tor.

Other researchers have argued, however, that the high success rates reported for WF attacks correspond to lab conditions that are unrealistically advantageous for the adversary compared to a deployment of the attack in the wild, and thus that the reported lab effectiveness is not informative of the practical effectiveness one can expect for actual Tor users [15]. A study by Juarez et al. [16] showed that the accuracy of WF attacks indeed drops when one considers scenarios with variations in parameters out of the control of the adversary but likely to occur in practice, such as: loading multiple pages in browser tabs; different versions of the Tor browser; webpage personalisation and staleness; or simply differences in the path of the Tor circuit through which the pages are downloaded. While the study does not dismiss WF attacks as impractical, it argues that effective defenses may be feasible with bandwidth and latency overheads that are much lower than what is required in artificial lab conditions.

Furthermore, we note that any delays introduced by a defense are critical for low-latency systems like Tor, as they have a direct impact on their usability for interactive applications. Heavy bandwidth overheads may also impact 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 edges, 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 II, we give a background of existing attacks and defenses, and discuss the suitability of the defenses for an implementation in Tor. In Section III, we provide a description of the system model considered in this paper and list the requirements that we believe a WF defense in Tor must fulfill.

A framework for the systematic implementation and evaluation of link-padding WF defenses. We have developed a framework to evaluate WF resilience of Tor pluggable transports. In Section IV 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 as soon as this work is published.

A lightweight defense against WF attacks. We have adapted Adaptive Padding, a traffic analysis countermeasure that was originally proposed to prevent end-to-end correlation attacks, to combat WF in Tor. We have dubbed this new defense *Website Traffic Fingerprinting Protection with Adaptive Defense* (WTF-PAD). Section V gives a specification of it, and in Section VI, we present an evaluation and a comparison of WTF-PAD with existing WF defenses.

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 [16], but they did not evaluate the effectiveness of WF defenses in these scenarios. In Section VII, we evaluate the effectiveness of WTF-PAD in the following two scenarios: (i) the *open-world*, in which the attacker monitors a small set of web pages and, (ii) the *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.

II. 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.

A. Attacks

The setting of the basic WF attack is as follows: a user browses pages in a web site through an encrypted channel (e.g., SSL/TLS, SSH, VPN, Tor), such that the specific webpage she is viewing remains private. The attacker is a *local* and *passive* adversary who eavesdrops traffic at some point in the network where the user can be identified (e.g., the IP of the client is visible). The adversary also has access to a set of webpages of interest, can download them through an equivalent encrypted channel, and can build templates of the network traffic they generate. We also assume that the attacker cannot break the encryption algorithms employed to protect the communication contents. Instead, the adversary identifies the pages downloaded by the user by analyzing the time and sizes of the network packets generated by the visit.

Typically, the WF problem is treated as a supervised classification problem, where the possible classification categories are URLs, and network *traffic traces*, namely sequences of packet inter-arrival times and packet lengths, are observations or *instances*. To deploy the attack, the adversary first trains a classifier with network traffic traces collected from his own visits to a set of target web pages he later wants to identify and link to users. Next, the attacker records and classifies user traffic and uses it to make a guess of the page that has been accessed.

The adversary can have different motivations to deploy a WF attack. For example, the objective of the attacker might be to identify specific pages within one single website. The first WF attacks succeeded in doing so over SSL connections [17], [18], and in 2010, Chen et al. reported high accuracy in inferring fine-grained interactions between a user and a website [19]. However, these attacks are not possible in connections with intermediate IP-stripping proxies (including Tor) or CDNs.

The most studied type of WF attack, and the one we tackle in this paper, considers an adversary who does not know the destination IP. In this case, the objective of the attacker is to identify pages that could be hosted in any (unknown) domain. In the early 2000s, two articles tackled this problem, first over one-hop web proxies [20] and later VPNs [21]. The authors of these studies 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.

It was not until 2009 that the first WF attack against Tor was published. Herrmann et al. presented an attack that achieved 3% accuracy with a Naive Bayes classifier [3] in a closed world of 775 pages 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, and state-of-the-art versions of the attack attain over 90% accuracy [4], [8], [5], [6], [7]. 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 [7]. Our own findings with their classifier (see Section VI), however, indicate that these open-world findings may be incorrect.

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, or running a Tor guard node. With the continuous improvement in WF classifier accuracy over the past few years, this is a pressing concern.

The methodology used to evaluate these attacks, however, has been criticized because it rests on assumptions that significantly favor the adversary and are unlikely to hold in practice [15], [22], [16]. For example, if we consider the vast size of the Web, and that the target set pages represents a small fraction of the total web traffic, then, the prior of a target page being visited is very low. It has been shown that in this case, due to the base rate fallacy, even classifiers with high true positive rates and low false positive rates fail in their predictions.

B. 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 Tor, 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.

High-level defenses. First, we focus on high-level defenses that work primarily based on information at the application layer. *HTTPOS* uses different techniques such as modifying HTTP headers and injecting HTTP requests strategically [10]. *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], [6], [7], [16].

Supersequence approaches and traffic morphing. In the last year, two studies proposed and analyzed a new set of defenses based on generalizing web page traffic traces [7], [12]. They create anonymity sets by clustering pages and defining their centroid as the minimum sequence that contains all the other sequences in the cluster. Next, they modify traces of pages falling in the same cluster to make them look like the corresponding centroid. The intuition behind this is that they optimally reduce the amount of padding needed to confound the attacker's classifier. These defenses, as well as traffic morphing techniques [9], [23], have the shortcoming that they require a large database of webpage templates that needs to be frequently updated and redistributed over the network. Distributing such information over the Tor network is challenging, as current distribution mechanisms such as those used to distribute the consensus do not scale [24].

Low-level defenses. Low-level defenses are designed to operate primarily at the network layer. In 2012, Dyer et al. extensively evaluated the impact of padding individual packets on the efficacy of the state-of-the-art attacks [8]. They evaluated a number of packet-padding strategies, ranging from a uniform padding to traffic-morphing techniques. The conclusion of this study was that, even though packet padding perturbs the packet-length distribution, it is not sufficient to hide coarse-grained features such as *bursts* in traffic or the total size and load time of the page. They did not evaluate these countermeasures in Tor, as 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 modified Ethernet traffic traces to make the transmission look constant-rate and with fixed-size packets. Their goal was to find an upper-bound for the security provided by an arbitrary link-padding-based countermeasure. However, the authors reported excessive bandwidth overheads in exchange 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 [14] and *CS-BuFLO* [5], [13], both based on BuFLO, attempt to optimize the original design. 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 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. Cai et al. also sketched CS-BuFLO as a practical version of BuFLO, extended with congestion sensitivity and rate adaptation. Following the same approach as Tamaraw of grouping pages in anonymity sets by 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 use in 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%. Additionally, 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.

III. System Model

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

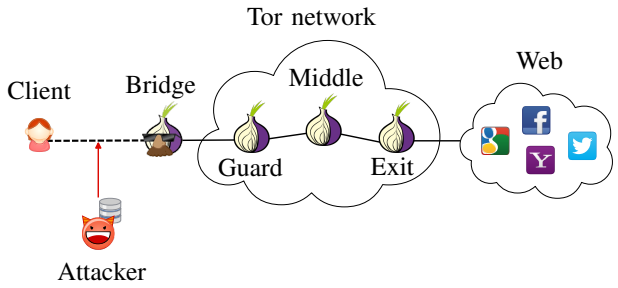


Fig. 1: The WF adversary model considering Tor bridges.

A. 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 of other parts of the network, and *passive*, meaning that it observes and records packets but does not modify, delay, drop or inject packets. The model is equivalent for a client connecting directly to Tor without a bridge, but the bridge-based model fits our framework.

We also assume perfect security of the cryptosystems used in the underlying onion routing protocol: the adversary cannot learn anything about packet payloads. However, the headers of the TCP, IP and link protocols are not encrypted and thus still visible.

In the attack considered in this paper, the adversary's objective is to determine whether the client is downloading one of a small set of target pages, that we call *monitored pages*. There is a much larger set of possible pages that the client could be visiting. This kind of open world setting fits a surveillance-like adversary that keeps a black-list of pages for which it wishes to deanonymize connections. 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.

B. Defense Model

Padding is performed end-to-end between trusted endpoints, with the adversary have 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*[1] (PT), avoiding modifications in the Onion Router source code.

To protect against malicious bridges, padding should be exchanged between the client and the guard. If malicious guards are also part of the threat model, then the padding should be sent between the client and the middle node.

We choose to implement our WF framework and countermeasures as PTs for several reasons. For the moment, the practicality of WF attacks against Tor users remains unclear, making it hard to justify the large-scale deployment of WF countermeasures with potentially heavy overheads that would significantly impact the quality and usability of the network as a whole. Implementing WF defenses as PTs instead 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. 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[2]. Since we are implementing padding as a PT, we require a framing layer that includes this flag in its header. In Figure 2, we 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.

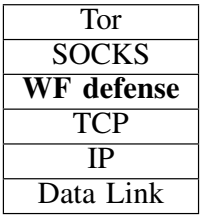


Fig. 2: The network stack of WF defenses implemented as Tor pluggable transports.

We also note that most PTs are designed to circumvent advanced traffic analysis techniques such as deep packet inspection [25], and thus aim to prevent a different, yet complementary, threat than that of WF.

C. Defense Requirements

We argue that a defense against WF in Tor should satisfy the following requirements:

1) *Effectiveness*. The defense must be effective. Namely, it must confound classifiers by reducing inter-class variance and increasing the intra-class variance of page instances. We argue that a defense need only provide this protection in a realistic setting, though with some margin for improvements in attacks.

2) *Usability*. The defense must not significantly impact the user experience. Ideally, it should be integrated in Tor transparently to the user. The delay introduced by link-padding should be minimized. A moderate bandwidth overhead should be acceptable to users for typical web browsing on broadband or 4G connections, but high bandwidth overheads could slow down browsing as well.

3) *Efficiency*. The defense should not use excessive bandwidth, as the costs to the mostly volunteer Tor router operators would make running the system prohibitive.

4) *No server-side cooperation*. The defense should not require cooperation from the web server, as the incentives of the servers to cooperate are little if any.

5) *No databases*. The defense should be implemented with out requiring a large database profiling many websites. Such a database is difficult to collect and must be updated regularly due to frequent changes in websites. Moreover, distributing this information to the clients poses technical challenges, as the current distribution mechanisms in the Tor network do not scale well [24].

[1] https://torproject.org/docs/pluggable-transports.html

[2] https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt

IV. Framework

Obfsproxy is the basis of most existing PTs and is available in several programming languages. 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 same application launches a sniffer that intercepts the traffic and records traffic measurements defined by the researcher. The main class describes an experiment, which we use to replay traffic traces over Tor or generate new traffic traces. This abstract class can be extended and used for the evaluation of new link-padding proposals. The framework is compatible with any transport based on Obfsproxy, and thus, it can be used to evaluate a broad range of traffic analysis defenses.

The framework allows for the collection of traffic traces as protected by a WF defense in three different modes. First, it allows the researchers to crawl new traffic traces over Tor using a PT that runs the defense. Web pages are constantly changing, and datsets of traffic data stale very fast. With this framework, one can obtain fresh datasets already protected with the PT defense installed. Second, the researcher might have collected traces over plain Tor and wish to apply the transformation of the defense without having to crawl again. The users of the framework can obtain protected traces from Tor traffic data locally. The last mode is to simulate the defense applying the transformation on existing traces without making any use of the network. This mode is even faster than the previous two but requires to have captures of data as collected at the application layer. The researchers might want to use this method for theoretical research as the simulations do not use the defense implemented in the PT.

The framework comprises the following modules or *experiment types*:

Crawler. The crawler collects the web traces over the PT and Tor. We can define "connection configurations" over which the crawler will connect. For instance, the basic setting used in this work is a connection over a Tor client and a bridge to the Tor network, but one could easily define a new connection which connects to the Tor network directly.

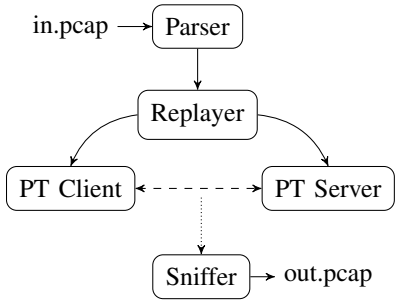


Fig. 3: Diagram of the replayer experiment.

Replayer. One of the main components of the framework is the replayer module, which is meant for reproducing previously collected web traces through the PT. The replayer reads the direction and inter-arrival time of each packet in the trace. For each packet, it strips its TCP payload and sends it to the corresponding endpoint (either the client or the relay), in order to simulate the direction of the packet as it was captured in the wire. We have manually verified that the timing and sizes of replayed packets are not noticeably different between padding and non-padding packets, since otherwise an adversary could remove the padding and recover the real payload.

Simulator. The simulator applies transformations to an existing TCP stream. It allows us 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 thus is mainly intended for theoretical research.

V. Adaptive Padding

There is a debate on whether probabilistic or deterministic strategies should be deployed to counter WF attacks [7]. 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 [26], can be adapted to protecting against WF because of 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 key features of the patterns — "*statistically unlikely*" delays between packets. Shmatikov and Wang showed that with a 50% padding rate, the accuracy of end-to-end timing-based traffic analysis is significantly degraded.

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 pre-determined 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.

Adaptive Padding (AP) works differently. It does not delay application data; rather, it sends it immediately. This minimal latency overhead makes AP very well suited for Tor. Further, AP pads the gaps between data packets so that the inter-arrival packet timings follow a certain distribution, rather than imposing a constant rate.

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 a being able to build a data set to train on for each website of interest, but he has the disadvantage of not being able to use precise timing information very reliably. Further, WF classifiers have advanced substantially since the introduction of AP in 2006.

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

A. 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. We also follow the terminology used in the specification of PTs, where *downstream* is the stream that flows between the defense endpoints and *upstream* is the traffic that flows from the application to the defense and from the defense to the web server (see Figure 4).

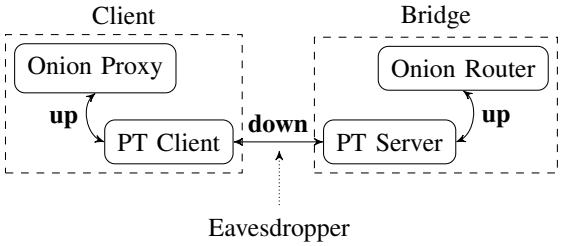


Fig. 4: The downstream is the stream between the PT client and the PT server and the upstreams are the streams that are not visible to the attacker.

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 so as to prevent long gaps from being a distinguishing feature. All padding follows from the real distributions of web traffic. Shmatikov and Wang recognized the importance of bursts in web traffic and thus developed a dual-mode algorithm to account for this. 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 that there is a gap between bursts and consequently aims to add a fake burst of padding with short delays between packets.

We note that this definition of a burst is different from that found in the WF literature, where a burst is typically defined as a sequence of consecutive packets in the same direction (inbound or outbound). In this paper, we follow Shmatikov and Wang and define a burst in terms of bandwidth. In particular, a burst is a sequence of packets that has been sent in a relatively short time. Conversely, a gap is a sequence of packets that have been sent over a longer timespan. Then, the objective of AP can be described as adding padding in low-bandwidth periods such that bursts are not as prominent and distinguishable.

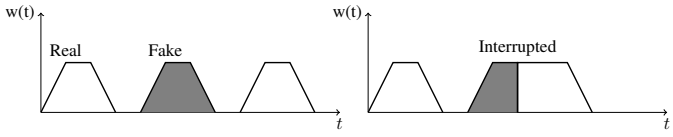


Fig. 3: A fake burst between two real bursts (left) and a fake burst interrupted by a real burst (right). The y-axis is bandwidth.

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 [4], [8], [27], 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. Also, burst mode may randomly switch to gap mode in the middle of a burst, leading to a fake burst in the middle of a real burst (see Figure 3). 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 for each time the page loads*. This reduces the utility of training the classifier with these features.

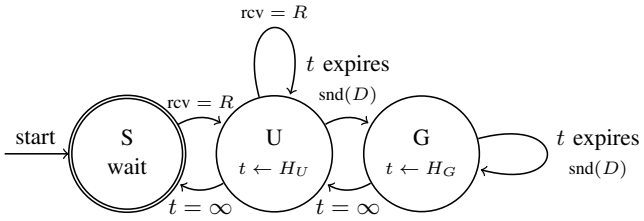


Fig. 4: Adaptive Padding finite state machine. AP instances in both ends follow this state machine. Receive events (*rcv* in the diagram) at the PT client refer to data being pushed from the application and receive events at the server refer to data coming from the web server.

AP algorithm. The basic AP algorithm is defined by two histograms of delays that we call H_U (used in burst mode) and H_G (used in gap mode). Let us consider traffic flowing in one direction only. As depicted in Figure 4, AP starts idle (state S) until transmission starts and it receives a real packet (R). This causes it to enter burst mode (state U), in which a delay t is sampled from the H_U histogram, and AP starts to count down until either a real packet arrives, or t expires. In the first case, 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). In state G, AP samples from histogram H_G and sends dummy messages when the times it samples expire.

An infinite delay can be sampled from the histograms. When an infinite delay is sampled while being in state G we jump back to state U. Similarly, a transition to S occurs when we sample an infinite time in U. Optionally, a transition from G to U can also occur upon receiving a real packet. We have one of these state machines in each defense endpoint.

A histogram is defined as a partition of the support of the distribution of times $[0, +\infty)$ into bins. Each bin is a half-closed interval $B_i = [a_i, b_i)$, with $0 \leq a_i, b_i \leq \infty$ for all $i = 1, \ldots, n$, where $n \geq 2$ is the number of bins. The partitions used in AP are distributed exponentially, namely, the intermediate bins have the following endpoints:

$$a_i = \frac{M}{2^{n-i}}, b_i = \frac{M}{2^{n-i-1}},$$

for $i = 2, \ldots, n-1$,where $M > 0$ is the maximum time considered in practice. The first bin is $B_1 = [0, \frac{M}{2^{n-2}})$ and the last bin is $B_n = [M, \infty)$.

An exponential scale for the bins provides more resolution for short times and less resolution for long times, which is very useful for distributions of inter-arrival times, which are heavily skewed towards short values.

When we sample from a bin, AP either interpolates or discretizes the associated interval. If we choose to interpolate, AP returns a value sampled uniformly from $[a_i, b_i)$. In case we discretize, AP returns b_i. In either case, when we fall into the bin $[M, \infty)$, we always return "∞".

Each bin contains a number of tokens k_i, and K is the total number of tokens in the histogram. Then, the probability of sampling a delay from that bin can be estimated as $P_i := \frac{k_i}{K}$. After sampling a time from a bin, a token is removed from that bin. If a real packet arrives, the sampled token is returned and a token is removed 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 recent transmission rate [26].

Burst mode. The H_U histogram governs how AP reacts to bursts during the transmission. It contains the times between the end of a burst and the beginning of the following burst for a large sample of sites. While we are in a burst, the delays we sample from H_U 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.

The distribution of tokens in H_U defines the probability of generating a false burst while in a gap, or adding a short false burst inside a real one. P_n is the probability of falling into the infinity bin and thus defines the probability of not sending padding (and thus generating a fake burst) when we draw a sample. We have set the number of tokens in B_n proportionally to the sum of the tokens in the rest of the bins:

$$k_n = P_n \sum_{i=1}^{n-1} \frac{k_i}{1 - P_n}.$$

For instance, if we set a 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 $k_n \approx 34$.

Gap mode. The histogram for gap mode, H_G, is built with the inter-arrival times *within* a burst in traffic collected for a large sample of sites. The distribution of times in H_G defines the length of a fake burst. The event of drawing a token from B_n after a certain number of messages follows a geometric distribution. Then, derived from the expected value of the geometric distribution, the expected number of messages sent until we fall into the infinity bin is $\frac{1}{P_n}$. Since fake bursts must have the same average length as real bursts, we set the expected number of samples until we hit the infinity bin to be the average burst length, that is $L_U = \frac{1}{P_n}$. Then, from this equation we can derive the number of tokens that we need to set in the infinity bin as a function on the number of tokens in the rest of bins:

$$k_n = \sum_{i=1}^{n-1} \frac{k_i}{L_U - 1}.$$

B. 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 4, where the "rcv" events in the transitions refer to downstream messages, instead of upstream messages. This allows to it 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 infinite bin in the burst state. This means that the distribution in H_U also governs the amount of padding appended to the end of the page. The probability of sending a sequence of false bursts follows a geometric distribution with parameter P_n. The number of tokens left in B_n depends on the number of fake bursts that have been added during the transmission. 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.

C. Interarrival time distribution

Shmatikov and Wang did not specify in the original AP paper how to build and use the distribution of interarrival times in the AP histograms. In their simulations, they sampled the interarrival times for both the real traffic and the padding from the same distribution. We need to set the distribution of dummy messages to be the same as the real messages in order to ensure that dummy and real messages are indistinguishable.

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 4000 pages and studied the distribution of interarrival times within their traces.

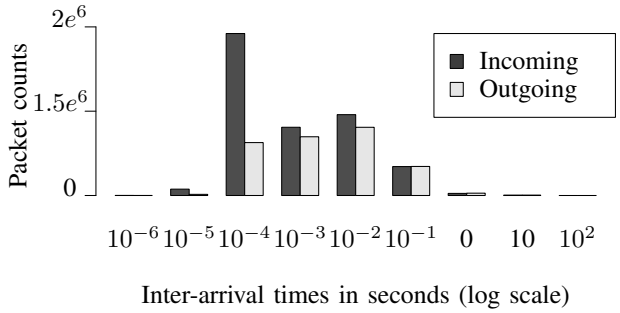


Fig. 5: Histogram of the inter-arrival time in a large sample of the top 35K Alexa websites.

In order to implement WTF-AP without revealing distinguishing features between real and fake messages, we need to send dummies in times intervals that follow the same distribution that real messages. In Figure 5, 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-AP has different histograms in the client and the relay we can simulate traffic that follows different distributions depending on the direction.

In our dataset, we observed that around 50% of the inter-arrival times are less than 1 millisecond. This presents an implementation challenge, since dummy messages must be sent with a precision of the order of microseconds, or we need to add sufficient noise to make the difference negligible to the attacker.

Another problem stems from the fact that these distributions depend on the connection of the client. For this reason, we cannot estimate these distributions a priori and ship them with WTF-PAD. This approach would have the same limitations as the clustering-based defenses that we have discussed in Section II. A solution we propose is to have WTF-PAD bootstrap the parameters of the distributions during the installation by making a series of connections to a set of pages. These parameters depend on the bandwidth and the state of the network and will need to be updated periodically. The update is performed by the client who would send distributions to the relay handling the defense, so the relays do not need their own measurements.

Next, we will need to find the bursts and the gaps in the inter-arrival times collected during the bootstrap phase and build the histograms H_U and H_G. Intuitively, burst-mode histogram H_U should consist of larger delays covering the duration of typical bursts, while 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 estimate the instantaneous bandwidth at the time of each inter-arrival period to determine if we are in a burst or not. Let $w(t)$ be an function that outputs the bandwidth being used at instant of time t. We need to set a threshold T, so that if $w(t) > T$ we know the interarrival times belong to a burst; otherwise, they belong to a gap.

In order to estimate $w(t)$, a common approach is to define a sliding window h over time. Then, $w(t) = \frac{N}{t-h}$, where h is the size of the time window and N is the number of bytes transmitted within h. We encountered the issue that for small values of h, this estimation suffers of *aliasing*. A sliding window over a sequence of consecutive packets is a more robust estimator. We have experimented with different number of packets as a window length and different thresholds. The best results against the k-NN attack are achieved for a window of two consecutive packets and T set to the total average bandwidth for the whole sample of pages.

D. Tuning bandwidth overhead versus security

The main limitation of AP is that, even though it can hide interarrival times that are longer than the average, it does not hide times that are shorter than the average. The only way to hide these times is either by adding delays or by adding more padding. In this section, we focus on the latter because our objective is to minimize delay. WTF-PAD provides a mechanism to tune the tradeoff 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.

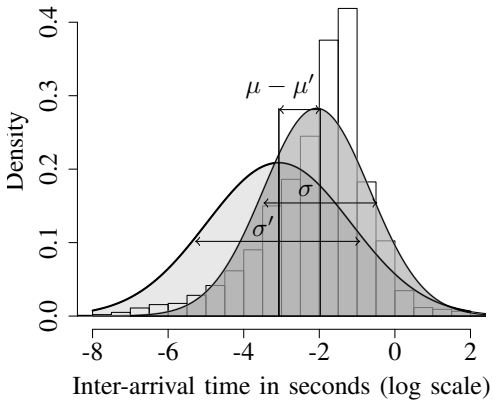


Fig. 6: Histogram of times between two consecutive bursts in the incoming direction. In dark gray we superpose the probability density function of our log-normal fit. In light gray, we show the density function of a shifted log-normal distribution that we use to build the H_U histogram.

In order to illustrate this, in Figure 6 we show the bins of the H_U 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 interarrival times follow a log-normal distribution. We

can modify the parameters μ and σ to obtain a new normal distribution $\mathcal{N}(\mu', \sigma'^2)$ that we will use to sample the inter-arrival times of H_U. We can do this transformation in the log domain as normal distributions. Then, the equivalent log-normal distribution will be defined using the same μ' and σ'.

In the figure, the region under the curve of the probability density function (pdf) of $\mathcal{N}(\mu', \sigma'^2)$ that does not overlap with the region enclosed by the pdf of $\mathcal{N}(\mu, \sigma^2)$ indicates the times that will be removed by AP in the distribution observed by the attacker. That is, we shift the average distribution of inter-arrival times toward shorter values. This results in more 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 interarrival times from the samples we have extracted from our dataset. Modeling network traffic is a well-known problem studied in the literature on network performance optimization. A notable result is that burstiness in web traffic holds a property called *self-similarity* caused by long-range dependencies between traffic events [28]. It turns out that this property is hard to model and working on this problem is out of the scope of this paper. For this reason, we relaxed our model and approximated the underlying distributions with normal and log-normal distributions.

To calibrate the possible shifts, we set μ' and σ' according to the percentile of the real data we want to include. For instance, assuming a normal distribution, if we adjust μ' to the 50th percentile, we obtain $\mu' = \mu$ and $\sigma' = \sigma$. If we set μ' to the value of the pdf at the 10th percentile, we then derive the σ' using the formula of the pdf of the normal distribution. Note that we can use the same method for log-normal distribution, simply using the log of the data instead.

This tuning mechanism is applied only on H_U, as it defines the probability of injecting padding inside bursts as well as appending a false burst after a real burst. At some point, we observe that by decreasing the mean interarrival time in H_U we are blurring the distinction between bursts. Since times in H_U are shorter than the times between two consecutive bursts, instead of adding a fake burst, WTF-PAD is appending padding to real bursts. This has a negative effect on the protection provided (see Section VI), limiting the amount of extra protection that can be added through this tuning mechanism.

VI. Evaluation

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

A. Data

Unlike most prior work, which used simulated data, we have used web traffic that has been collected over Tor. Our datasets were collected for a related study about a realistic evaluation of WF attacks [16]. In their work, several variables were studied 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 [29]. 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.

B. Methodology

To evaluate the improvements in performance offered by the defense, we compared the original traffic traces with traces that have been protected by applying the defense. We used the framework described in Section IV to simulate WTF-PAD on the traces of our dataset. The difference in bandwidth between the original trace and the protected trace provide 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 attack is based on a k-NN model that includes more than 4000 features [7]. 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.

The state-of-the-art attack is based on a k-NN model. k-NN is a supervised learning method, meaning that it constructs hypotheses on data that have been labeled beforehand. When we feed the classifier with a test instance, k-NN measures the distance from that instance to all training instances. Next, it selects the k closest instances, also called the neighbors, and assigns a class depending on the votes of the neighbors. In particular, the distance used by Wang et al. is a weighted sum of a set of features. This feature set is the most extensive in the WF literature and includes fine-grained features that exploit traffic bursts. Some of these features are constructed based on little 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 [14]. 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

TABLE I: Performance and security comparison among link-padding defenses against the most effective attacks in the closed-world setting. For CS-BuFLO, BuFLO and Tamaraw, we either used the values they have reported or applied their implementations for the attacks they had not evaluated. The parameters of the defenses have been chosen to find reasonable bandwidth-versus-security trade-offs.

		Accuracy (%)				Overhead (%)	
Defense	Parameters	kNN	Pa-SVM [4]	DL-SVM [6]	VNG++ [8]	Latency	Bandwidth
BuFLO [8]	$\tau = 10$s, $\rho = 20$ms, $d = 1500$B	14.9	14.1	18.75	N/A	145	348
CS-BuFLO [13]	$\rho = [20, 200]$ms, $d = 1500$B, CPSP	N/A	30.6	40.5	22.5	173	130
Tamaraw [7]	$\rho_{out} = 0.053$, $\rho_{in} = 0.138$, $d = 1500$B	13.6	10.59	18.60	12.1	200	38
WTF-PAD	Normal fit, $p = 0.4$, $d = 1500$B	17.25	15.33	23	26	0	54

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.

We also evaluated the defense with other classification models that take into account features that we consider interesting and are not included in the set of features of k-NN.

C. Results

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

In Figure 7, we show the results of this comparison using different statistical models. We experimented with multiple positively skewed distributions such as *Pareto, Weibull, Gamma, and Beta* to build our models and used the Kolmogorov-Smirnov test to evaluate the goodness of fit. We estimated the parameters of the distributions using maximum likelihood estimation. Even though Pareto and Beta distributions seemed to best fit our inter-arrival time distributions, we decided to use normal and log-normal distributions for the sake of simplicity, given that the error was not significantly greater than that observed in the other distributions.

The consequences of the mismatch between the model and the true distribution can be seen in Figure 7, where we show the bandwidth overhead ratio versus the accuracy obtained by the classifier. For instance, we observe a steeper decrease in accuracy for the normal model with respect to the log-normal. This difference is due to the amount of inter-arrival times that are discarded because have much less probability in our model.

The effect of the interruptions of fake bursts by real bursts can be observed at the bottom of the graph. WTF-PAD reaches a point in which the inter-arrival times are so short that fake and real gaps merge and the defense is no longer effective. This results in either excessive overhead or a increase in the success of the attack.

In Table I, we summarize the security versus overhead tradeoffs obtained for different attacks (i.e., kNN, NB, SVM, DL) and defenses BuFLO, Tamaraw, CS-BuFLO and WTF-PAD. For sake of comparison, we used values reported in the past and completed those evaluations by applying new combinations of attacks and defenses using the data available.

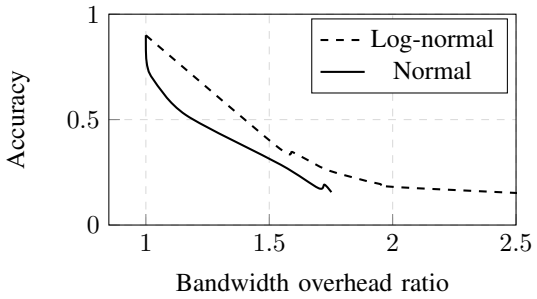


Fig. 7: Average accuracy versus average bandwidth overhead ratio.

As we see, WTF-PAD is the only defense to provide zero latency overhead.[3] 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.

VII. 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.

A. 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 developed a new methodology to evaluate the open world for k-NN, as we believe there are two serious flaws in the evaluation setting of Wang et al. First, they measured the error of the k-NN model to predicting instances that have not been seen before using leave-one-out cross-validation. In leave-one-out cross-validation, only one instance of the data

[3]This is in simulation. The increase in bandwidth use could add moderate delays in practice, particularly for bandwidth-constrained clients.

is held out for testing, and the rest of the instances are used for training. In other words, if there are n instances in total, each instance in the whole open world dataset is used $n-1$ times for training and only once for testing. The large overlap of the training sets introduces a correlation among the models trained at each round of the leave-one-out, which increases the prediction error estimated by cross-validation [30]. This is commonly known as overfitting, as the models reduce the training error but increase the variance of the error in the cross-validation estimates of the classifier performance to new data. For a large n and given the large feature set used in the k-NN attack, the prediction error caused by overfitting can be critical, as the models are more complex and try to explain more variables in the training set. Thus, in the open-world, where n is potentially infinite, such an evaluation may give optimistic results on how the classifier behaves in large sizes of the open world.

The second issue is related to the number of instances used for training. Wang et al. considered only one instance for each page in the training set for the unmonitored pages, which introduces error in the training set that is not due to the classifier – the classifier tries to explain variables in the data that are erroneous. However, it is fair to think the attacker will provide the classifier with several examples of pages from both the monitored and non-monitored sets.

In order to address the first issue, we used *k-fold* cross-validation. k-fold cross-validation for large values of k also has variance in the prediction estimate. In fact, leave-one-out is k-fold with $k = n$. The choice of k needs to be adjusted to find a compromise between the variance in the prediction of the model to new data and the bias of the model to our data. We have experimented with different values of k ranging from $k = 10$, a value recommended in most of the applied machine learning literature, to lower values such as $k = 4$, which will decrease the variance at expense of a higher error rate in the model prediction. We tackle the second issue above by using 40 instances for each of the training pages.

In the worst case for the user, the features of the monitored pages are distinctive within the set of all possible pages, as the classifier can easily discriminate them from the non-monitored set. We ranked the pages in the closed world by the success rate of the k-NN attack in correctly classifying them and selected different ranges of the Alexa Top 100 to build our training set. Here, our motivation is to simulate an open world as realistically as possible. It is not known whether there is a relation between the prior of the page and the distinctiveness of its features, so we decided to take the best case for the adversary.

Going against the attacker in a realistic setting, however, is the base rate fallacy. In short, when classification has even a modest rate of false positives, and when there are many non-monitored pages that users are viewing, any session detected as being a monitored page becomes more likely to be a false positive on a visit to a non-monitored site. The open-world setting includes a small number of monitored pages and a much larger number of non-monitored pages. Also, we did not try to represent the percentage of traffic these monitored pages receive in reality. We considered the scenario in which all pages have the same popularity, and hence for large world size, the monitored pages have a low probability of being visited.

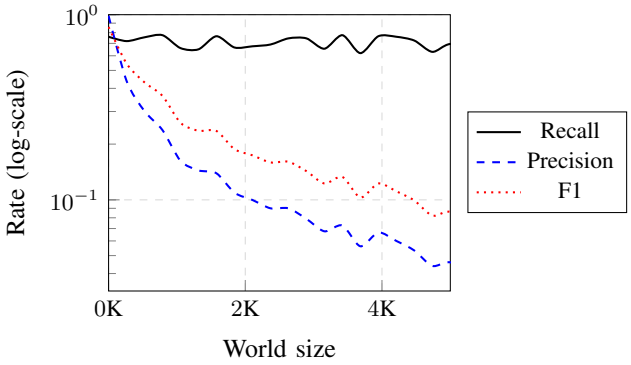


Fig. 8: 10-fold cross-validated performance metrics of the classifier for different sizes of the open world and a size of the monitored set of 50 web pages. The data point at size zero corresponds to the value of the metrics in a closed world of 100 pages.

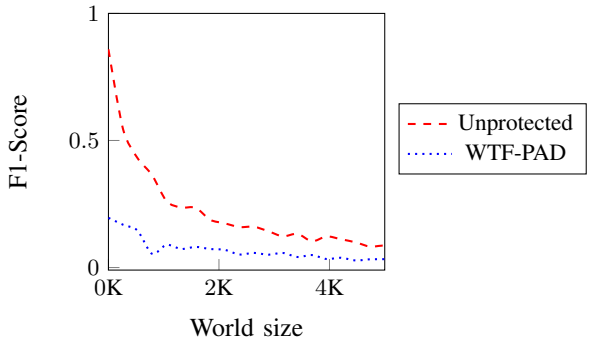


Fig. 9: Mean F1-Score with and without defense for different world sizes.

In Figure 8, we see the F1-Score, Recall and Precision for an open world of up to 5,000 web pages. We observe that the FPR tends to 0.14 for a large size of the world. It remains roughly constant because the k-NN attack does not lose accuracy on detecting non-monitored pages. However, if we look at the absolute number of false positives, an adversary who monitors a world of 363 pages, will incorrectly classify 266 non-monitored instances as monitored and, for a size of 5,000 pages, it commits 4,324 false positive misclassifications. The effect of false positives in a big open world is reflected by the Precision and F1-Score metrics, whose value drops off dramatically to 0.007 and 0.16, respectively.

We also evaluated the open-world setting with traces that had been protected with WTF-PAD. We plotted the F1-Score using the traces with and without protection. A strategic adversary who knows that the defense is in place will train the classifier using instances protected with the defense. As we see in Figure 9, the low accuracy obtained by the classifier for a small world of 100 pages (approximately 20%) rapidly drops when we consider a large size of the open world. Both values monotonically decrease until values that are below 1%. The F1-Score for a world of 35K without protection drops until

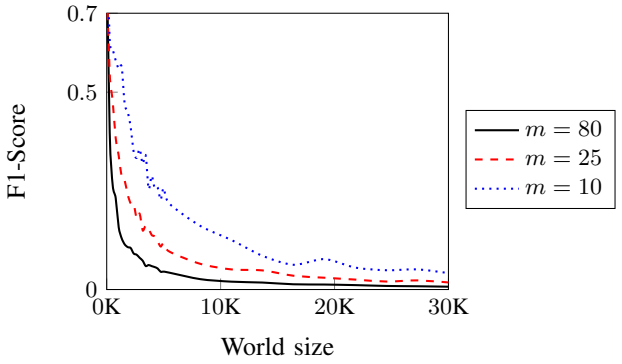


Fig. 10: F1-Score for different sizes of the monitored set (m).

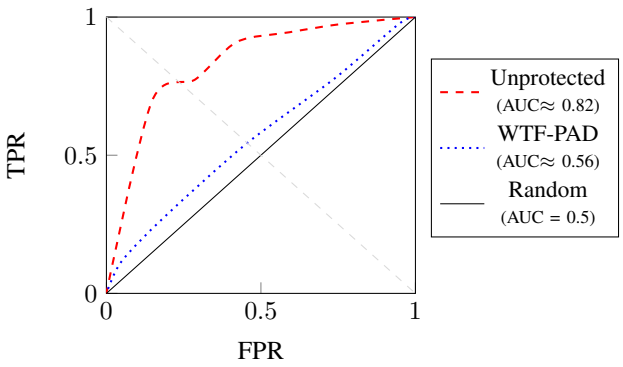


Fig. 11: Mean ROC curves of the k-NN classifier with $k = 5$ neighbors and using a consensus threshold. Each data point represents a different value for the minimum number of votes required to reach an agreement, ranging from 1 (only one vote to the positive class is sufficient to classify it as monitored) to 5 (all neighbours must agree). The size of the world is 5,000 pages and there are 50 monitored pages.

0.016, and the F1-Score for the protected traces decreases until 0.005.

Size of the monitored set. The proportion of the monitored set with respect to the size of the world might have a positive effect on the success rate of the classifier in the open world. We have observed that a smaller monitored set of pages boosts the performance of the classifier for small sizes of the world. However, this does not hold for larger sizes of the open world. For all the monitored set sizes we have evaluated (i.e., 10, 25, 80), we observe a steep decrease in the first 10,000 pages and then asymptotically tend to values that are below 1%.

ROC curves. In order to study the impact of WTF-PAD on the trade-off between FPR and TPR of the classifier, we also plotted the ROC curve for the data with and without the defense. k-NN is not a parametric algorithm, meaning that there is no explicit parameter that we could 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.

To plot the ROC curve we experimented with different number of neighbours k. The curves in Figure 11 are the curves with greater area under the curve (AUC) for all the values ok k that we tested, which was calculated using the trapezoidal rule on the data points that we had obtained. We notice a significant reduction in the performance of the classifier with the set of protected traces. Compared to the ROC curve of the k-NN on the unprotected data, the ROC of k-NN with WTF-PAD applied on the data is substantially closer to random guessing.

B. Multi-tab evaluation

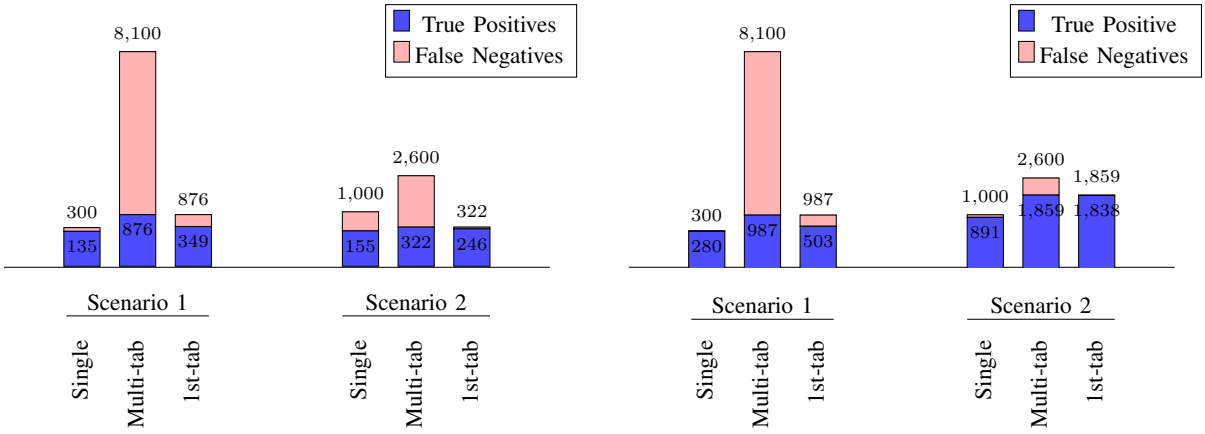
In this section, we assume the victim is using Tor for web browsing, with no other applications running over Tor, but she may open multiple tabs and windows that will establish multiple concurrent HTTP sessions. Opening multiple tabs is a common practice, and even though a given tab may seem to be finished 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. [16], 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.

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 list[4]. In the evaluation of this scenario, we used 10-fold cross-validation to suppress the effect of overfitting. Table II 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 II), the defense does not impact significantly the TPR of the classifier. As shown in Figure 12a, 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

[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.



(a) With WTF-PAD

(b) Without WTF-PAD

Fig. 12: The bars represent the True Positives and False Negatives for each traffic type in the test set (the number of instances of each type of traffic in the test set is the number at the top of the bar). The third bar shows the number of background pages (the first tab) detected among truly detected multi-tab traces.

TABLE II: True Positive Rates for protected and unprotected traces in Scenarios 1 and 2.

	TPR	
	Protected	Unprotected
Scenario 1	12%	15%
Scenario 2	13%	76%

protection, k-NN successfully detects them with 93% accuracy (i.e., Pr$[TP \mid$ single-tab$] = 0.93$), which is roughly the TPR that k-NN achieves in the closed-world scenario. The accuracy in detecting single tab traces, that is, Pr$[TP \mid$ single-tab$]$, protected by WTF-PAD is 45%.

To evaluate Scenario 2, we trained and tested k-NN on a dataset that includes multi-tab and single-tab traces. We show the distribution of each type of traffic in the test set in Figure 12b. In contract to Scenario 1, Scenario 2 is used exactly using the same set of data, for this reason we opted for using the leave-one-out cross-validation implemented in the original k-NN implementation. We note that in the multitab setting we evaluate a small set of pages (100 pages), thereby are not concerned about the variance introduced by leave-one-out in the open world.

Table II shows the accuracy of the k-NN classifier in Scenario 2. In this scenario, the attack achieves 76% TPR on unprotected multi-tab traces, much higher than the 15% found in Scenario 1. However, the success rate on protected traces drops to 13%. Figure 12b 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 89%, which is close to the accuracy rate of k-NN in the closed-world setting. The accuracy decreases to just 15% 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. If the delay is high, such as five seconds, before loading the foreground page, then the start of the foreground page is usually free from much intereference by the background page.

VIII. Discussion

In this paper we present evidence that shows that a probabilistic defense is less expensive in terms of latency and bandwidth overhead than defenses based on constant-rate padding and provides similar levels security. In realistic scenarios, particularly the open-world setting, a moderate level of security is likely sufficient to make website fingerprinting impractical. In this section, we discuss whether WTF-PAD is potentially suitable for Tor, open research questions that should be addressed in WF generally, and ways that WTF-PAD can be improved.

A. Suitability for Tor

Website fingerprinting is a significant threat to the security of Tor users. WF attacks fall within the Tor threat model [2], 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 [16],

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 limiting 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) [31]. To the extent that this holds today, only the remaining 60% of traffic needs to be covered by this defense. Second, the key 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. 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.

To illustrate the availability of bandwidth in Tor, consider statistics from April to June 2015. Approximately 30 Gbps out of the 90 Gbps of advertised guard bandwidth was consumed, while approximately 7 Gbps out of the 130 Gbps of advertised middle node bandwidth was consumed.[5] Thus, there is ample guard and middle-node bandwidth to handle a moderate overhead for link padding defenses.

B. Open Issues in Realistic Scenarios

Although 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 users are going to on Tor 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 have significant drawbacks, unfortunately, and remain a part of future work.

[5]These estimates are somewhat inaccurate due to relays with middle and exit flags being double counted.

C. Limitations of WTF-PAD

The construction of the histograms H_U and H_G 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* [32]. We think that developing more sophisticated models of network inter-arrival times can improve the performance of WTF-PAD.

Second, the distribution of inter-arrival times depends on the average bandwidth of the client. We have built our statistical model based on the inter-arrival times observed in our dataset. However, AP lacks a systematic method to set the optimal configuration for each client. We propose to have WTF-PAD bootstrap these values during installation. In the first run, we can make a series of downloads of a template content to estimate the bandwidth and tune the histograms accordingly. The statistics might be kept on the client safely, as they do not reveal anything about the user browsing history.

Future work in developing and implementing WTF-PAD should further explore the tuning of the histograms. A possible idea is to utilize an evolutionary strategy to find the optimal histogram for each specific situation. A genetic algorithm could optimize a fitness function composed by the bandwidth overhead and the accuracy of the WF attack. Under mild assumptions on the distribution, histograms can be represented efficiently to reduce the search space.

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.

IX. Conclusion

In this paper, we described the design of WTF-PAD, a probabilistic link-padding defense based on Adaptive Padding, modified to work with Tor and address a number of realistic issues required for deployment. 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 a significant reduction on bandwidth consumption of AP compared to BuFLO, Tamaraw and CS-BuFLO. Most importantly, WTF-PAD does not introduce any delay in the communication, which is especially important 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, even though the defense performance is comparable to other defenses in the closed-world, the security provided by any defense in these situations is greater than expected.

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] Tor project, "Users statistics," https://metrics.torproject.org/users.html, (accessed: July 20, 2015).

[2] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *USENIX Security Symposium*. USENIX Association, 2004.

[3] D. Herrmann, R. Wendolsky, and H. Federrath, "Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier," in *ACM Workshop on Cloud Computing Security*. ACM, 2009, pp. 31–42.

[4] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website Fingerprinting in Onion Routing Based Anonymization Networks," in *ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2011, pp. 103–114.

[5] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a Distance: Website Fingerprinting Attacks and Defenses," in *ACM Conference on Computer and Communications Security (CCS)*, 2012, pp. 605–616.

[6] T. Wang and I. Goldberg, "Improved Website Fingerprinting on Tor," in *ACM Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2013, pp. 201–212.

[7] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective Attacks and Provable Defenses for Website Fingerprinting," in *USENIX Security Symposium*. USENIX Association, 2014, pp. 143–157.

[8] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2012, pp. 332–346.

[9] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *Network & Distributed System Security Symposium (NDSS)*, 2009.

[10] X. Luo, P. Zhou, E. Chan, and W. Lee, "HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows," in *Network & Distributed System Security Symposium (NDSS)*. IEEE Computer Society, 2011.

[11] M. Perry, "Experimental Defense for Website Traffic Fingerprinting," Tor project Blog. "https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting", 2011, (accessed: October 10, 2013).

[12] X. Cai, R. Nithyanand, and R. Johnson, "Glove: A Bespoke Website Fingerprinting Defense," in *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2014, pp. 131–134.

[13] ——, "CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense," in *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2014, pp. 121–130.

[14] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses," in *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2014, pp. 227–238.

[15] M. Perry, "A Critique of Website Traffic Fingerprinting Attacks," Tor project Blog. "https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks", 2013, (accessed: December 15, 2013).

[16] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A Critical Analysis of Website Fingerprinting Attacks," in *ACM Conference on Computer and Communications Security (CCS)*. ACM, 2014, pp. 263–274.

[17] H. Cheng and R. Avnur, "Traffic Analysis of SSL Encrypted Web Browsing," *Project paper, University of Berkeley*, 1998, Available at http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps.

[18] S. Mistry and B. Raman, "Quantifying Traffic Analysis of Encrypted Web-Browsing," *Project paper, University of Berkeley*, 1998, Available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.5823&rep=rep1&type=pdf.

[19] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," pp. 191–206, 2010.

[20] A. Hintz, "Fingerprinting Websites Using Traffic Analysis," in *Privacy Enhancing Technologies (PETs)*. Springer, 2003, pp. 171–178.

[21] Q. Sun, D. R. Simon, and Y. M. Wang, "Statistical Identification of Encrypted Web Browsing Traffic," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2002, pp. 19–30.

[22] T. Wang and I. Goldberg, "Poster: Practical Website Fingerprinting on Tor," in *IEEE Symposium on Security and Privacy (S&P))*. IEEE, 2014, http://www.ieee-security.org/TC/SP2014/posters/WANGT.pdf.

[23] L. Lu, E. Chang, and M. Chan, "Website Fingerprinting and Identification Using Ordered Feature Sequences," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2010, pp. 199–214.

[24] J. McLachlan, A. Tran, N. Hopper, and Y. Kim, "Scalable onion routing with torsk," in *Proceedings of ACM CCS*, 2009, pp. 590–599.

[25] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The Parrot Is Dead: Observing Unobservable Network Communications," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2013, pp. 65–79.

[26] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," *European Symposium on Research in Computer Security (ESORICS)*, 2006.

[27] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, "I know why you went to the clinic: Risks and realization of https traffic analysis," in *Privacy Enhancing Technologies*. Springer, 2014, pp. 143–163.

[28] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835–846, 1997.

[29] Alexa, "Alexa Top 500 Global Site," http://www.alexa.com/topsites, 2015.

[30] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2, 1995, pp. 1137–1145.

[31] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker, "Shining light in dark places: Understanding the Tor network," in *Privacy Enhancing Technologies Symposium (PETS)*, July 2008.

[32] M. Zukerman, T. D. Neame, and R. G. Addie, "Internet traffic modeling and future technology implications," in *Annual Joint Conference of the IEEE Computer and Communications (INFOCOM)*, vol. 1. IEEE, 2003, pp. 587–596.