Comparing Website Fingerprinting Attacks and Defenses

Tao Wang
Cheriton School of Computer Science
University of Waterloo

Ian Goldberg
Cheriton School of Computer Science
University of Waterloo

***Abstract*—Website fingerprinting attacks allow a local, passive eavesdropper to identify a web browsing client's destination web page by extracting noticeable and unique features from her traffic. Such attacks magnify the gap between privacy and security — a client who encrypts her communication traffic may still have her browsing behaviour exposed to low-cost eavesdropping. Previous authors have shown that privacy-sensitive clients who use anonymity technologies such as Tor are susceptible to website fingerprinting attacks, and some attacks have been shown to outperform others in specific experimental conditions. However, as these attacks differ in data collection, feature extraction and experimental setup, they cannot be compared directly. On the other side of the coin, proposed website fingerprinting defenses (countermeasures) are generally designed and tested only against specific attacks. Some defenses have been shown to fail against more advanced attacks, and it is unclear which defenses would be effective against all attacks. In this paper, we propose a feature-based comparative methodology that allows us to systematize attacks and defenses in order to compare them. We analyze attacks for their sensitivity to different packet sequence features, and analyze the effect of proposed defenses on these features by measuring whether or not the features are hidden. If a defense fails to hide a feature that an attack is sensitive to, then the defense will not work against this attack. Using this methodology, we propose a new network layer defense that can more effectively hide all of the features we consider.**

I. Introduction

In order to protect their privacy, Internet users may want their web browsing activities to be hidden from curious eavesdroppers. To hide their destinations, these users must, at the least, encrypt their communication traffic and obscure their destinations with a proxy. *Website fingerprinting* refers to the set of techniques that seek to re-identify these clients' destination web pages by passively observing their communication traffic. The traffic will contain features such as unique packet lengths, packet length frequencies, packet ordering, and interpacket timings. A number of attacks have been proposed that would compromise a client's expected privacy, while defenses have been proposed to counter these attacks.

Website fingerprinting is a threat to any client who may expect her browsing behaviour to be monitored. Even users of Tor, an anonymization network, are vulnerable to website fingerprinting attacks [3], [19], [26]. Understanding the true impact of website fingerprinting attacks and potential defenses is imperative to protecting the privacy of its estimated 500,000 daily users [24], as well as users of SSH tunneling, VPNs, IPsec, and other privacy-enhancing technologies. In this paper, we will investigate the impact of website fingerprinting attacks and defenses under different adversarial scenarios.

Some website fingerprinting attacks achieve higher accuracy rates than others, but it is not clear why. In addition, incomplete experimental techniques, small data sizes, strong assumptions, and the fickleness of Internet traffic dumps have placed the realistic effectiveness of some of these attacks in doubt [21]. A comparison between different attacks, beyond a simple comparison of accuracy rates, is lacking. Possible barriers to an even comparison include differing attack scenarios, contrary assumptions, and difficulties in running large-scale experiments. For instance, training a classifier to distinguish between 1000 web pages (with 100 instances each) will require a computation time on the order of 10^9 CPU seconds for one of the latest attacks [26], and collecting this data set for Tor would require approximately $4 \cdot 10^6$ seconds on a single connection. These factors make it difficult to understand why certain attacks outperform others, and what criteria a successful attack should satisfy.

On the other hand, proposed website fingerprinting defenses are always evaluated against only a few attacks. Often, the evaluation further assumes that the attacker is not aware of the defense. In 2012, Dyer et al. [7] evaluated a number of website fingerprinting defenses and showed that they were ineffective against three particular website fingerprinting attacks [12], [15], [19]. It remained unclear, however, *why* defenses failed or succeeded, and deciding whether a defense is worth its overhead has been difficult for Tor developers [20]. Our methodology allows us to explain these results.

In this paper, we present a feature-based comparative methodology that aims to address the above problems. We go further than Dyer et al. and survey a set of nine attacks and six defenses in the literature. Additionally, we expose the features that each attack focuses on by comparing them on packet sequences that only differ by a single feature. This allows us to explain the relationship between different attacks by offering a unified basis for comparison. We measure the sensitivity of each attack to our features; an attack that is highly sensitive to a feature that distinguishes web pages will be effective, while an attack that is highly

sensitive to a feature that is not useful in distinguishing web pages will be ineffective and easily misled. Similarly, we evaluate whether each defense is able to hide these features: defenses that successfully hide a feature will be effective against attacks that rely on that feature. We present these defenses while estimating their overhead in bandwidth and in time taken to load a page by analyzing real traffic. This allows us to explain why certain defenses have proven to be ineffective in the past, and what can be done to improve them.

The contributions of this paper are as follows:

1) We present a new feature-based comparative methodology, described in Section III, which offers a high-level intuition into the workings of each proposed website fingerprinting attack and defense. Such a comparison has been lacking in the literature. We implement and test all of the attacks and defenses on the same platform and same data set, and we present the results in Sections IV and V.
2) Using the above methodology, we gain new insights and are able to answer several key questions in the literature. In Section IV-B, we are able to pinpoint the reasons behind the success or failure of different attacks in various scenarios. In Section V-B, we compare and classify defenses, expose some of their flaws, as well as answer questions posed by Tor developers on their applicability to Tor. To help privacy-sensitive users choose a suitable defense, we present the overhead of each defense in Section V-C.
3) We present a new defense, `Tamaraw`, in Section VI that hides all of the features we identified more successfully than any of the surveyed defenses. We do so by optimizing the BuFLO defense from Dyer et al. [7] to reduce unnecessary overhead and fix a gap in its defensive properties.

II. Fundamentals

Website fingerprinting (WF) is the problem of analyzing the traffic of a web browsing client to identify her destination web server, even when the client is using secure encryption through a proxy to hide this information. WF is a subproblem of traffic analysis in general, which is a diverse set of methods used to extract different types of information from communications traffic. A *WF attack* is a set of techniques used to expose the client's web browsing behaviour, involving data collection, processing, feature extraction, and classification; a *WF defense* is a set of countermeasures used to protect the client against a WF attack by obfuscating, or *covering*, features of the web traffic. When a feature is covered, it becomes less distinguishable across different classes of interest; for example, across different web pages. This reduces the accuracy of the attack if the classifier depends on this feature. In this section we discuss the background of WF. We first describe different attacker scenarios and client scenarios found in the literature, and then describe how a web page is loaded and how this allows fingerprinting to occur.

A. Attacker scenario

The attacker in our scenario is a curious eavesdropper who wishes to identify which web page the client is loading. Cai et al. [3] and Chen et al. [5] describe variants wherein an attacker aims to identify a web site instead of a web page; for consistency across the literature, however, we focus on the identification of single web pages. The attacker must possess some method of observation. For example, he could have wiretapped the client's connection, or he could control some of the intermediate routers. For each page load the attacker observes its *packet sequence*, which is an ordered list of packet lengths and their corresponding times. The contents of the packets are assumed to be encrypted and yield no useful information. The client uses a proxy to obfuscate her destination; the proxy is at or after the attacker's tap. Luo et al. [18] give an overview of possible attack vectors for various technologies. We show the respective locations of the attacker and proxy in Figure 1.

The attacker is assumed to be local and passive: the attacker taps and observes from only one location, and he is not allowed to add, drop, or change packets. This means that the attacker is weak, but is also resource-light and essentially undetectable. Active attacks such as packet spoofing [10], remote ping detection [11] and measuring memory fingerprints [14] may achieve the same goals as WF attacks; however, we do not consider these attacks here, as they all use different types of information and involve vulnerabilities or assumptions of vulnerabilities which do not fall under our systematization.

We retain two assumptions that all previous works on WF have made of the attacker. First, the attacker is able to notice exactly when a new page is requested. In other words, the attacker knows which sequence of packets corresponds to a single page. This assumption is sometimes made in a stronger form — that the client's think time always dominates the page load time — which implies that a distinct pause exists between any two packet sequences of different page loads. Second, any background activity the client may have will not interfere with the client's traffic. For example, a client will not download a file while visiting a web page; alternatively, these file download packets can be easily discarded by the attacker. These assumptions are used by all previous works on WF as they simplify the problem, though it should be noted that these assumptions are advantageous for the attacker.

Client interaction with a web page (e.g. AJAX) is outside the scope of this work; we refer the reader to previous works on attacking or defending against interaction-based fingerprinting [4], [5].

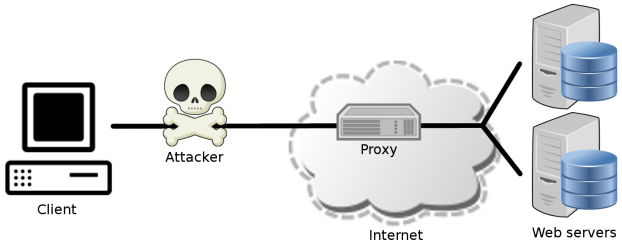

Figure 1. Diagram indicating the attacker's location.

Before the attack, the attacker gathers a training set by visiting the pages that the attacker is interested in detecting (mimicking the client's setup) and collecting the packet sequences. The training set is used to train a classifier, which learns how to decide which page each testing packet sequence belongs to. The training set is often assumed to be up to date, but at least one work [26] has explicitly used training sets that are somewhat out of date to take into account the high training time of the attacker's classifier. In a *closed-world* attack scenario, the attacker trains on a limited set of pages, and the simulated client is only allowed to visit those pages; the attacker's goal is to determine which page was visited. In an *open-world* attack scenario, the client can visit any page, and the attacker's goal is to determine whether that page was one of particular interest.

B. Client scenario

While the attacker scenarios in the works we survey are similar, the assumptions regarding the client's setting differ significantly. Attacks against one client setting can be ineffective against another. Roughly speaking, the assumptions regarding the client can be classified into three categories, in terms of the amount of information leaked to the attacker:

1. Resource lengths. To load a web page, a client must load each constituent *resource* of that web page, such as images and scripts. In this setting, the attacker is able to observe the length of each resource on a web page. This information is not usually available; however, it may be leaked, for example, if different concurrent connections can be distinguished from each other, in which case the packets between each GET request correspond to a single resource [23]. This strong assumption was used by the earliest WF attacks [6], [13], [23]. As it is not known how to extract individual resource lengths from within modern protection technologies, such as SSH tunneling, VPNs, IPsec, and Tor, this setting has become inapplicable, and we do not implement any of the attacks under this setting.

2. Packet lengths. In this setting, the attacker is able to observe the length of each packet. This is the more common WF setting, and corresponds to privacy-enhancing technologies such as SSH tunneling, VPNs, and IPsec over a proxy. These technologies, by default, make no extra effort to obscure packet lengths or timings, and most attacks under this setting focus on uniquely identifiable packet lengths, whereas defenses attempt to cover them. This setting is strictly more difficult to fingerprint than the first setting.

3. Obscured packet lengths. In this setting, fixed-length packets are sent so that packet lengths will not uniquely identify a web page. This corresponds roughly to Tor's setting; with Tor, content is delivered in fixed-size 512-byte cells, a variable number of which are then sent together in SSL records. The newest attacks are focused on this setting, which is strictly more difficult than the first two, so attacks that achieve success under the second setting may prove ineffective in this setting [12].

None of the WF attacks we survey assume that the client obscures packet timing, so we will assume it is revealed to the attacker.

The client is assumed to have disabled caching, although Cai et al. have done some work for the case when the client has cached the destination web page [3]. This assumption is justified on the Tor Browser, for example, which disables disk caching. Enabling caching may allow an active timing attack where the attacker spoofs a request to a web page in order to determine if the client has visited it before [8].

C. Loading a web page

When loading a web page, the first resource requested by the client is the main HTML page. The main page usually requests other resources, such as scripts, images, text, or other components of the web page. Upon being notified of each required resource, the client generally tries to request the resource as soon as possible.

Different resources on a web page may be fetched from different servers. For each resource, the client opens (persistent) HTTP connections to its server. After the connection is established, the client sends a GET request for the resource. The number of permitted parallel HTTP connections to the same server and to different servers can be different for each browser and each browser version, with the recent trend being an upwards increase. For example, in Firefox 3, the maximum number of connections to any one server is 2, and the maximum number of connections overall is 40; in Firefox 23.0, they are 6 and 256 respectively. Upon receiving a new resource GET request, Firefox attempts, in order, to find the first idle connection and either reuse (if the servers are the same) or drop it; open a new connection if the connection limit is not reached; or place the request in a queue. Other browsers may handle this logic differently.

Whenever a connection is idle or closed, a request from the request queue can be sent. At this time, the browser may instead choose to send out multiple requests at the same time for resources located at the same server — without waiting for the resources to arrive in between. This is known as pipelining. Pipelining saves round-trip times, but it must be supported by the end server. Pipelining is currently disabled by default on Firefox and Chrome; however, it is enabled by default on the Firefox distribution in the Tor Browser

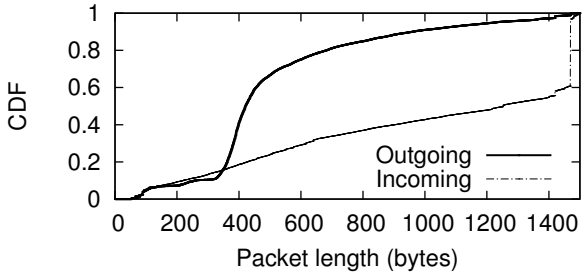

Figure 2. Packet lengths observed when loading one instance of each of Alexa's top 800 sites. Packets sized 1500 bytes are discarded.

Bundle, as part of an experimental defense against WF (see Section V-A).

We can see that the order of requests is logically deterministic, induced by the structure of the web page (the list of resources, their lengths, and which resources each resource requests). If there is only one connection, then the sequence of packet lengths (but not their timings) can be predetermined, allowing easy website fingerprinting. However, the interaction of multiple connections causes randomization in packet ordering. Two connections opened to the same server would often handle different subsets of the resources required to load a page each time the page is loaded. The difficulty is compounded by the fact that the resources of most web pages vary, as many pages have randomized content such as advertisements. A successful WF attack will therefore tolerate these randomized differences while learning to distinguish different web pages.

III. Features and our methodology

A classifier succeeds at distinguishing between two classes when it is able to discover a consistent difference between them. This can be viewed as a difference between their features, which characterize a class. Implicitly or explicitly, classification techniques such as WF attacks extract features to classify. Conversely, a successful defense covers these features. In this section, we describe how our methodology leverages features to allow us to compare WF attacks and defenses.

A. Features

In general, packet sequences have four major features.

The first is unique packet lengths. Packet lengths are a simple and strong feature of a web page. GET request lengths are partly determined by the length of the resource name. Incoming packets are almost always sent at the Maximum Transmission Unit (MTU), with the length of the last packet indicating the size of the resource (modulo the MTU). Most packet lengths of a page are unlikely to change unless resource lengths change. WF attacks almost always consider packet lengths unless they are designed for the Tor scenario, in which case packet lengths are covered. When unspecified, we assume that packet lengths are not hidden from the attacker, but we address the scenario where they are and the algorithms that continue to work under this scenario. To identify the distribution of unique packet lengths, we visited Alexa's top 800 sites [1] once each and present the distribution in Figure 2, excluding the approximately 75% of packets that were sized 1500 bytes. Outgoing packets are concentrated in the 400–600 byte range, whereas incoming packets are relatively evenly distributed with a spike at 1470 bytes, which was the MTU of some of our connections. TCP and IP header lengths sometimes change over the course of page loading, which in turn changes the amount of data inside an MTU packet and thus affects the resulting unique packet lengths.

The second major feature is packet length frequency. Packet length frequency is the number of times each packet length occurs. The number of incoming packets at MTU size is a rough estimation of the total size of the page, which changes due to random advertisements and updated content. Most of the earlier WF attacks explicitly discard packet length frequencies [12], [15], [17]. We will show that all current WF defenses fail to cover the total traffic size, as doing so is difficult and would necessarily incur a large traffic overhead.

The third major feature is packet ordering. The structure of a page induces a logical order onto its packet sequence. As an example, a GET packet for a resource can only be sent once the reference to that resource is received by the client. The attacker may be able to infer information about the content of each packet from observing packet ordering. Packet ordering depends on network conditions: it may vary due to bandwidth and latency, and it may be affected by changing the parameters for persistent HTTP connections, pipelining, and so on. Tor developers have implemented a prototype defense based on packet ordering by randomizing request order (see Section V-A).

The fourth major feature is interpacket timing, which reveals the logical relationship between packets. For example, viewing from the client's end, the outgoing server connection SYN and the incoming SYNACK will differ by a round-trip time; so will the GET request and the first packet of that resource. If the attacker's tap is near the client, then a short inter-packet time between an incoming packet and a following outgoing packet suggests that the outgoing packet could be caused by the incoming packet. On Tor, the client's circuit would have different latency, congestion, and bandwidth values from those the attacker trained on, making it more difficult for the attacker to use interpacket timing.

A packet sequence P can be written as:

$$P = \langle (t_1, \ell_1), (t_2, \ell_2), ..., (t_n, \ell_n) \rangle$$

In the above, t_i is the difference in time observed between packets i and $i-1$ (interpacket timing), with $t_1 = 0$; ℓ_i is the byte length of packet i. The sequence length, $|P|$, is equal to n. We write P_t and P_ℓ as the sequences of only the

interpacket times and only the packet lengths, respectively. We indicate the packet length as a positive value if the packet is outgoing and as a negative value if it is incoming.

We define the four features mathematically as follows.

Unique packet lengths. P and P' are said to have different unique packet lengths if their sets of packet lengths are different:

$$(\exists L \in P_\ell | L \notin P'_\ell) \vee (\exists L \in P'_\ell | L \notin P_\ell)$$

Packet length frequencies. Suppose $n_L(P_\ell)$ is the number of times packet length L appears in P_ℓ. P and P' are different under this feature if their packet lengths occur at different frequencies. However, packet lengths that are unique to either P or P' are not considered here:

$$\exists L | n_L(P_\ell) \neq n_L(P'_\ell) \wedge n_L(P_\ell) > 0 \wedge n_L(P'_\ell) > 0$$

Packet ordering. We denote M_ℓ as the multiset of packet lengths in P_ℓ, without ordering. We say that two sequences P and P' have different packet ordering if:

$$M_\ell = M'_\ell \wedge P_\ell \neq P'_\ell$$

Interpacket timing. Suppose P and P' have sequence lengths $|P|$ and $|P'|$. P and P' are said to have different interpacket timings if their timings are different:

$$\exists i, 1 \leq i \leq \min(|P|, |P'|) : (P_t)_i \neq (P'_t)_i$$

We next discuss some properties of our feature set.

Fact 1. *If packet sequences P and P' are not the same, then they must differ in at least one of the four features above.*

This fact demonstrates that our choice of features is, in some sense, complete, in that it represents any difference between two packet sequences. We can therefore claim that successful attacks should expose at least one of those four features between packet sequences, while defenses are effective if they can cover all four features.

Fact 2. *Given any packet sequence P and any subset of the above features, there exists a packet sequence P' such that P and P' only differ in this subset of features, with the exception that different packet ordering implies that unique packet lengths and packet length frequencies do not differ.*

This fact implies that our features are somewhat independent of each other (except packet ordering). It should therefore be possible to find generators that change one specific feature without affecting the others, allowing us to pinpoint which features various attacks depend on, and which features various defenses attempt to cover.

B. Comparative methodology

We describe our feature-based comparative methodology as follows. To determine if an attack is able to extract a feature, we apply the attack to two classes, C and C', which differ only by that feature. We use a generator G to transform C into C' by causing a change in some feature of each packet sequence $P \in C$ and inserting the output into C'. We parameterize $G^{(v)}$ by v, a non-negative integer such that the greater the value, the more "different" P and $P' = G^{(v)}(P)$ will be; we require $G^{(0)}(P) = P$. We design each generator to modify only one specific feature.

$G^{(v)}$ operates from the start of the packet sequence. Informally, $G^{(v)}$ is equivalent to $G^{(1)}$ repeated v times, possibly from different starting points in the packet sequence. For all of our generators, this interpretation ensures that v functions as a magnitude.

Our generators are not randomized. We designed each generator to accept values of v up to $|P|/5$; the maximum value of v used in our experiments is 180. None of the generators produce a packet length greater than the MTU.

We give a textual description of each generator below and explicitly define each generator $G^{(v)}$ in Table I.

Unique packet length generators.

1) Small packet length changes. All packet lengths are increased by v, up to MTU.
2) Large packet length changes. v packet lengths are increased by 1000, up to MTU.
3) Diffusing packet lengths. v packet lengths are increased by their position divided by 5, up to MTU.

Packet length frequency generators.

4) Appending incoming MTU packets. v incoming MTU packets are appended to the end.
5) Appending outgoing packets. v outgoing packets are appended to the end, their lengths being the lengths of the first outgoing packets of P.
6) Inserting incoming MTU packets. v incoming MTU packets are added, one per 5 packets.

Packet ordering generators.

7) Adjacent transpositions. v packets are transposed with the previous packet.
8) Short-distance transpositions. v packets are transposed with the packet 4 elements ago.
9) Long-distance transpositions. v packets are transposed with the packet 19 elements ago.

Interpacket timing generators.

10) Delays. Each packet is delayed by a linearly increasing amount of time, multiplied by v.

We chose these generators from our prior knowledge of how WF attacks and defenses behave, in order to highlight their differences.

Table I
GENERATORS. PACKET SEQUENCE $P = \{p_1, p_2, ..., p_N\}$ WHERE $p_i = (t_i, \ell_i)$, $t_1 = 0$. WITH P_{out} AS THE SEQUENCE OF OUTGOING PACKETS IN P, WE DEFINE P_{out_i} AS ITS i^{TH} ELEMENT (WRAPPING BACK TO THE BEGINNING IF $i > |P_{out}|$). d_i IS THE DIRECTION (1=OUTGOING, -1=INCOMING) OF p_i. $A(P, i, p)$ APPENDS PACKET p AFTER p_i. $T(P, i, j)$ TRANSPOSES THE PACKET LENGTHS OF p_i AND p_j.

Feature type	Generator name	#	Transformation for $G^{(v)}_{\#}$
Unique packet lengths	Small packet length changes	1	For $0 < i \leq \|P\|$: $\ell_i \leftarrow d_i \min(\|\ell_i\| + v, 1500)$
	Large packet length changes	2	For $0 < i \leq v$: $\ell_{5i} \leftarrow d_{5i} \min(\|\ell_{5i}\| + 1000, 1500)$
	Diffusing packet lengths	3	For $0 < i \leq v$: $\ell_{5i} \leftarrow d_{5i} \min(\|\ell_{5i}\| + i, 1500)$
Packet length frequencies	Appending incoming MTU packets	4	Repeat v times: $P \leftarrow A(P, N, (t_N, -1500))$
	Appending outgoing packets	5	For $0 < i \leq v$: $P \leftarrow A(P, N, (t_N, P_{out_i}))$
	Inserting incoming MTU packets	6	For $0 < i \leq v$: $P \leftarrow A(P, 5i, (t_{5i}, -1500))$
Packet ordering	Adjacent transpositions	7	For $0 < i \leq v$: $T(P, 5i, 5i - 1)$
	Short distance transpositions	8	For $0 < i \leq v$: $T(P, 5i, 5i - 4)$
	Long distance transpositions	9	For $0 < i \leq \lfloor v/5 \rfloor$, $0 < j \leq 5$: $T(P, 25i - j, 25i - 19 - j)$
Interpacket timing	Delays	10	$\forall p_i \in P, t_i \leftarrow t_i + v \cdot i \cdot 0.020\,\text{ms}$

C. Data Collection and Experimental setup

Our attacker trains a classifier on labeled training instances, and tests the classifier on testing instances for which the label is not known to the classifier; this is known as supervised machine learning. In this paper, we focus on distinguishing between two classes:

$$C = \{P_1, P_2, ..., P_{400}\}$$
$$C' = \{G^{(v)}(P_1), G^{(v)}(P_2), ..., G^{(v)}(P_{400})\}$$

These are the original class and the generator-modified class with one feature changed. Since our generators operate on packet sequences, the elements of C and C' are packet sequences. We use the first 200 elements for training and the last 200 elements for testing. The training and testing sets are denoted C_{train} and C_{test}, with the generator-modified sets being C'_{train} and C'_{test}.

We construct C by connecting to `bbc.co.uk` 400 times with caching disabled. The reason why we do so is as follows. C should contain packet sequences of the same page rather than different pages, because WF attack classifiers are designed to tolerate the randomness within the same page while exposing the differences between different pages. A successful classifier should therefore be able to distinguish C and C' despite the randomness in C (and therefore C'). On the other hand, the elements of C need to differ from each other; this will allow us to measure how sensitive each classifier is to the generators' operations. The page we chose has a suitable amount of randomness and has therefore been difficult to classify [26].

We extract the packet length of each TCP packet, using the packet length indicated in the IP total length field; this value includes the lengths of both the TCP and IP headers. We chose to include the lengths of these headers to help identify the MTU packets.

A class C may have a convenient classifier representation denoted as M_C; for example, M_C could be a list of the unique packet lengths in C, or the mean length of packet sequences in C. M_C is learned from C and used to test incoming packets.

Our experimental setup is as follows. We load pages over a 100 Mbps Ethernet connection with MTU set at 1500. Although our feature-based methodology allows us to use a smaller data set for training and testing, the computational time required was still high for several of the attacks. For automated page loading, we used iMacros 9.00 on Firefox 23.0. We collected data with `tcpdump` and parsed the data into packet sequences with our code. We implemented all of the attacks and simulated defenses in this paper in a combination of Python, C++, and C, and all of them are available upon request.

IV. WEBSITE FINGERPRINTING ATTACKS

We present the list of WF attacks we have surveyed in Section IV-A. We apply our feature-based comparative methodology to each of these attacks, and present the results in Section IV-B. Finally, we include a summary of the run time and storage costs of the attacks in Section IV-C.

A. List of attacks

Our list of attacks is given in chronological order. For each attack we assign a name by which it will be referred, starting with the first two letters of the first author's surname, followed by hyphen and then a descriptive phrase. We discuss implementation details of each attack to clarify our decisions where the original authors of the attack were not clear.

Resource length attacks

Some of the earliest WF attacks assumed that web resource lengths were revealed to the attacker, such as Cheng et al. in 1998 [6], Sun et al. in 2002 [23], and Hintz in 2003 [13]. In these works, web resource lengths are used as unique identifiers for a web page. Without the persistent connections offered by HTTP/1.1, older technologies such as SafeWeb [13] expose individual resource lengths to an attacker who can distinguish between connections, as the data transferred for a connection corresponds to one resource. Most modern privacy-enhancing technologies, such as SSH tunneling and Tor, do not leak resource lengths. As such, resource lengths are not assumed to be available to the attacker in all of the following works we will survey, and so we will not analyze schemes that require them with our methodology. However, the observation that unique resource

lengths are important will remain relevant in the form of unique packet lengths in overall packet sequences.

Cross correlation on timing and lengths (Bi-CrossCor)

Bissias et al. in 2006 published the only attack we are aware of that uses interpacket timings [2]. It does not rely on knowing the resource lengths of the web page. The classifier representation for class C in `Bi-CrossCor` is a sequence of interpacket times $M_C = \langle m_1, m_2, ..., m_N \rangle$, where N is the length of the longest packet sequence in class C, and m_i is the mean of the interpacket times of the i^{th} packet in each packet sequence (in order). The cross correlation between two ordered lists $S = \langle s_1, s_2, \ldots, s_n \rangle$ and $T = \langle t_1, t_2, \ldots, t_n \rangle$ is:

$$X(S,T) = \frac{\sum_{i=1}^{n}(s_i - \overline{S})(t_i - \overline{T})}{\sigma_S \sigma_T}$$

where $\overline{S}$, $\overline{T}$, σ_S and σ_T are the means and standard deviations of S and T respectively. To classify packet sequence P, we compute $X(M_C, P_t)$ for each class C, taking only the first $|P|$ interpacket times for M_C. We also carry out the same computation by replacing interpacket times with packet lengths (discarding the direction). Bissias et al. found that the best results are achieved by multiplying the two cross correlation values obtained for interpacket times and packet lengths.

Jaccard's coefficient on packet lengths (Li-Jaccard)

Liberatore and Levine in 2006 described two new WF attacks [15]. These attacks demonstrated that unique packet lengths help classify a web page. Packets are almost always sent at the MTU whenever possible, and therefore packet lengths which are not the MTU are indicative of the remainders of resource lengths. The first attack they proposed relies on the Jaccard coefficient. The classifier representation for class C is a set $M_C = \{\ell_1, \ell_2, ..., \ell_n\}$ of unique packet lengths for each packet sequence $P \in C$. A packet length ℓ_i is inserted into M_C if ℓ_i is present in the majority of packet sequences in C. The direction is included as a positive/negative sign on the packet lengths.

The Jaccard coefficient is a measurement of the similarity of two different sets (S and T) as follows:

$$J(S,T) = \frac{|S \cap T|}{|S \cup T|}$$

Two packet sequences are therefore more similar if they share more packet lengths. To classify P, we find the set of all packet lengths in P, denoted U_P. We then measure $J(M_C, U_P)$ for all classes C, and classify P as the highest scoring class.

`Li-Jaccard` discards packet timing, ordering, and frequency, keeping only packet lengths. Liberatore and Levine showed that it achieved a greater accuracy compared to `Bi-CrossCor`, which relied on more features, indicating that unique packet lengths are an important identifiable feature. However, `Li-Jaccard` had a lower accuracy than the same authors' second attack, described below.

Naïve Bayes (Li-NBayes)

The second attack proposed by Liberatore and Levine was a Naïve Bayes classifier based on packet lengths and their frequencies within a packet sequence. Suppose we want to decide if packet sequence P belongs to class C. We extract $S(P) = \{(\ell_1, f_{\ell_1}), (\ell_2, f_{\ell_2}), ..., (\ell_n, f_{\ell_n})\}$. Here, ℓ_i are packet lengths in P with direction and f_{ℓ_i} are the numbers of times they appear in P. The classifier representation M_C for class C is:

$$M_C = \{(\ell_1, F_{\ell_1}), (\ell_2, F_{\ell_2}), ..., (\ell_N, F_{\ell_N})\}$$

F_{ℓ_i} are multisets of all observed frequencies of that packet length in each sequence. That is to say, if there are $|C_{train}|$ training packet sequences, then each F_i will have length $|C_{train}|$. Then, the Naïve Bayes assumption says that we assume that the packet lengths and their frequencies (ℓ_i, f_{ℓ_i}) occur independently of each other, and the score assigned to $P \in C$ is:

$$\prod_{1 \le i \le |S(P)|} p(f_{\ell_i} \in F_{\ell_i})$$

The probability $p(f_{\ell_i} \in F_{\ell_i})$ can, for example, be computed with a normal kernel density estimation, that is to say,

$$p(f_{\ell_i} \in F_{\ell_i}) = \frac{1}{\sigma_i} e^{-\frac{(f_{\ell_i} - \mu_i)^2}{\sigma_i^2}}$$

In the above, μ_i and σ_i are the mean and standard deviation of F_{ℓ_i}. We set the minimum of the standard deviation σ_i to be 1.0, and the minimum of this term to be 10^{-100}, to apply Cromwell's rule: we never allow the probabilities to be zero.

Liberatore and Levine found that `Li-NBayes` trumped `Li-Jaccard` in almost every situation under their experimental setting; this may be because `Li-NBayes` keeps packet length frequencies whereas `Li-Jaccard` does not.

Multinomial Naïve Bayes (He-MNBayes)

Herrmann et al. proposed a number of improvements to Liberatore and Levine's scheme by incorporating techniques from text mining that are known to improve the accuracy of matching documents (web pages) when searching with keywords (unique packet lengths) [12]. They use the Multinomial Naïve Bayes classifier, with the term frequency (TF), inverse document frequency (IDF), and cosine normalization (CN) schemes.

The Multinomial Naïve Bayes classifier changes the computation of $p(f_i \in F_i)$. We denote S_i as the sum of all elements in F_i. Instead of using the normal kernel, we compute $p(f_i \in F_i)$ as follows:

$$p(f_i \in F_i) = \left(\frac{S_i}{\sum_{i=1}^{N} S_i}\right)^{f_i}$$

TF, IDF and CN modify f_i, that is, the exponent term, of the above formula. TF transforms the power logarithmically:

$$TF(f_i) = \log(f_i + 1)$$

IDF discards common packet lengths, magnifying the effect of unique packet lengths. We note that specifically, all incoming MTU packets will be discarded by IDF. Suppose $C_i \subseteq C_{train}$ is the set of training packet sequences that contain the packet length i. Then:

$$f_i^* = TF(f_i) \cdot \log \frac{|C_{train}|}{|C_i|}$$

CN is used to further modify the frequency by dividing the modified frequency with the Euclidean norm of all modified frequencies:

$$CN(f_i^*) = \frac{f_i^*}{||(f_1^*, f_2^*, ..., f_N^*)||}$$

All of these transformations are applied to the exponent term of $p(f_i \in F_i)$, such that it becomes $CN(f_i^*)$.

Herrmann et al. achieved a higher accuracy than Liberatore and Levine in comparable experiments. However, they also demonstrated that their attack is ineffective on Tor. Our feature-based comparative methodology will be able to explain these results.

Levenshtein distance on unique packet lengths (Lu-Levenshtein)

Wright et al. proposed a WF defense called traffic morphing (see Section V-A). In response, Lu et al. published an attack that is capable of distinguishing morphed packet sequences [17]. It does so by heavily focusing on packet ordering, choosing to explicitly discard packet length frequencies and packet timing. The attack relies on the observation that the order of non-MTU packets does not often change between packet sequences derived from the same web page. From each packet sequence P two sequences M_{out} and M_{inc} are extracted, where M_{out} are all the outgoing packet lengths in order and M_{inc} are all the incoming packet lengths in order, with MTU packets (size 1500) removed.

To classify a testing packet sequence P_{test}, we compare the similarity between the testing packet sequence and each training packet sequence P_{train}, and assign the testing packet sequence to the class of the most similar training packet sequence. Similarity is computed as:
$1-0.6\cdot D(M_{out_{test}}, M_{out_{train}})-0.4\cdot D(M_{inc_{test}}, M_{inc_{train}})$
D is the Levenshtein distance, which is equal to the number of insertions, deletions and substitutions of packet lengths to transform one packet sequence into another, and then normalized by the length of the longer sequence. This process is done separately for the incoming and outgoing packet sequences, and combined linearly with weights 0.6 and 0.4. Lu et al. chose these weights as they produced good results, and demonstrated that their attack is able to achieve WF at a high accuracy as well as defeat traffic morphing. The authors also pointed out the importance of the open-world scenario (which they called the problem of detection).

Features extraction on SVM (Pa-FeaturesSVM)

Panchenko et al. published an attack that specifically targeted web browsing clients that use Tor, as Herrmann et al. were not able to successfully perform WF on Tor [19]. On Tor, unique packet lengths are obscured by Tor's padding of traffic to fixed-size cells. Panchenko et al. therefore leveraged a number of other features, which are processed by a Support Vector Machine (SVM).[1] An SVM feature is either a number or a list of numbers; some of the features used by Panchenko et al. have a fixed length while others have variable lengths that are dependent on the packet sequence. A full list of features is given by Panchenko et al., and we implement them all. Features with fixed lengths are placed at the front of the list, followed by features with variable lengths, in the order presented by Panchenko et al.

Panchenko et al. demonstrated that this attack is powerful enough to achieve WF on Tor at an accuracy comparable to attacks on clients who are not using Tor, and they were the first to do so. With our implementation, we were not able to achieve WF using their cost parameter C and smoothness parameter γ. This is because the cost and smoothness parameters depend on the nature of the problem; ours is a two-class problem with controlled differences between the classes whereas theirs is a multi-class problem We decided to optimize `Pa-FeaturesSVM` on $G_6^{(5)}$, which is a suitable goal for the attack, by varying these parameters over powers of 2^5. We achieved the best results with $C = 2^{25}$, $\gamma = 2^{-45}$.

Damerau-Levenshtein distance (Ca-DLevenshtein)

Cai et al. improved the accuracy of WF on Tor by using the Levenshtein distance, but with transpositions included [3]. They called their algorithm the Damerau-Levenshtein distance[2]; an optimal algorithm for the computation of this distance is known in literature [16]. An SVM is trained by calculating the distances between all pairs of training packet sequences. Like `Lu-Levenshtein`, to classify a testing packet sequence P_{test}, `Ca-DLevenshtein` compares P_{test} with each training packet sequence P_{train}. The distance matrix replaces the kernel matrix normally derived from features. (More details on SVMs are given in the book of Vapnik and Chervonenkis [25].)

As Tor cells hide actual packet lengths, Cai et al. round each TCP packet length upwards to the nearest 600 in order to estimate the number of Tor cells transmitted. We did not change their parameters.

[1] Dyer et al. [7] later use a similar but smaller set of features for a variable n-gram classifier, but their classifier did not perform better in any of the scenarios they considered.

[2] The algorithm given by Cai et al. is more commonly known as the optimal string alignment distance.

Combined Optimal String Alignment Distance (Wa-OSAD)

Wang and Goldberg further improved the accuracy of Cai et al.'s scheme on Tor by incorporating a number of modifications into the optimal string alignment distance algorithm [26]. Before calculating the distance, Tor cells are extracted directly from the packet capture by reconstructing TCP streams and parsing the SSL records contained therein. The locations of the Tor SENDME control cells are guessed and they are removed from the sequence. As our generators operate on TCP packet lengths, our methodology does not support these methods, and we will therefore revert to Cai's round-by-600 scheme.

The other modifications made by Wang and Goldberg to the distance computation are as follows. They disable substitutions as these do not correspond to the variations caused by loading the same page repeatedly. They increase the cost for outgoing packet operations as it is less likely for the same page to have a different number of outgoing packets; this number is closely related to the total number of resources, after removing SENDMEs. They also increase the transposition cost near the beginning of the packet sequence than at the end. These modifications increase the accuracy of fingerprinting in both the closed-world and open-world scenarios. Similar to `Ca-DLevenshtein`, these distances, once computed, are fed into an SVM.

Fast Levenshtein-like distance (Wa-FLevenshtein)

Wang and Goldberg showed another attack with weaker accuracy than both `Ca-DLevenshtein` and `Wa-OSAD` under comparable experiments, but which was around three orders of magnitude faster to train and test. Rather than finding the least-cost sequence of insertions, deletions, and transpositions, the algorithm first performs the necessary number of deletions from the end, and follows up with transpositions. This reduces the time complexity from quadratic to linear in terms of the packet sequence sizes. The complete algorithm is given by Wang and Goldberg [26].

B. Results

We tested the nine attacks in Section IV-A against the ten generators in Table I. We present the results in Table II. For each generator $G^{(v)}$ we vary v from 1 to 180, and we present three figures: the minimum values of v such that for all greater values of v (up to 180), the attack achieved a higher accuracy than 0.55, 0.75, and 0.9 respectively. These values show us the sensitivity of the attack to the generator G, as v functions as the magnitude of the feature change. An attack that achieves a higher accuracy at lower values of v is more sensitive.

We highlight the following results:

1) Despite its lower accuracy, `Bi-CrossCor` is highly sensitive to packet ordering. This suggests that sensitivity to a feature does not translate directly into accuracy; in fact, as some features (such as packet ordering) change across multiple accesses of the same page, a successful classifier must also be sufficiently insensitive to changes *within* the class.
2) `He-MNBayes` is only sensitive to unique packet lengths, and not to packet length frequencies, ordering, or timing. As Tor covers unique packet lengths, this explains why `He-MNBayes` was shown to have a very low accuracy on Tor [12]. The superior sensitivity of `He-MNBayes` on the unique packet length generators compared to previous attacks may explain its success in distinguishing web pages without packet length obfuscation.
3) `Li-Jaccard`, `Li-NBayes`, `He-MNBayes` all discard packet order and interpacket timing, achieving no success on any of those generators. It is interesting to see that while at first glance `He-MNBayes` is a slight modification of `Li-NBayes`, `He-MNBayes` is not dependent on packet length frequencies at all, unlike `Li-NBayes`.
4) The accuracy of the four SVM attacks can be roughly seen from their sensitivities to the three packet length frequency generators: `Pa-FeaturesSVM` and `Wa-FLevenshtein` are lower, while `Ca-DLevenshtein` and `Wa-OSAD` are higher.

As a whole, we see that attacks generally became more sensitive to more features over time; `Bi-CrossCor` is an exception to this trend. Only `Bi-CrossCor` is sensitive to changes in interpacket timing (if it does not cause re-ordering). While attacks on systems that reveal packet lengths (`Li-Jaccard`, `Li-NBayes`, and `He-MNBayes`) ignored packet ordering, attacks designed for the Tor setting (`Pa-FeaturesSVM`, `Ca-DLevenshtein`, `Wa-OSAD` and `Wa-FLevenshtein`) use both packet ordering and packet length frequencies, which is necessary for accurate WF attacks on Tor.

C. Run time and storage costs

`Ca-DLevenshtein` and `Wa-OSAD` are the most expensive attacks to train. The training time was around 2 CPU hours on a set of 400 training packet sequences, compared to several seconds for all of the other attacks. Real-time classification of a testing packet sequence is possible for all of the attacks except `Lu-Levenshtein`, `Ca-DLevenshtein`, and `Wa-OSAD`, for which testing time is significant (as we need to compute the distance between the testing packet sequence and each training packet sequence). The storage required is much larger for `Ca-DLevenshtein`, `Wa-OSAD` and `Wa-FLevenshtein` than for other attacks (to store the entire set of distance comparisons), but it is not large. For these attacks, around 320 KB is enough for 400 training packet sequences; it increases with the square of the number of training sequences.

V. Website fingerprinting defenses

Having seen what features each website fingerprinting attack uses to distinguish web pages, we now turn our

Table II

RESULTS ON THE SENSITIVITY OF OUR ATTACKS TO FEATURES. FOR EACH GENERATOR, THE COLUMNS SHOW THE MINIMUM VALUES OF v FOR WHICH THE ATTACKER HAD A HIGHER ACCURACY THAN 0.55, 0.75 AND 0.9 RESPECTIVELY FOR GREATER VALUES OF v. THEY ARE MARKED WITH AN ASTERISK IF THE CORRESPONDING ACCURACY IS NOT REACHED FOR $v \leq 180$. HIGHER VALUES (AND ASTERISKS IN THE LIMIT) INDICATE LESS SENSITIVITY TO THE FEATURE IN QUESTION.

Attack	Unique packet length									Packet length frequency									Packet ordering									Timing		
(Chronological)	G_1			G_2			G_3			G_4			G_5			G_6			G_7			G_8			G_9			G_{10}		
Bi-CrossCor [2]	16	161	*	1	3	3	69	*	*	4	11	*	4	8	*	1	2	6	43	*	*	1	1	47	4	4	19	95	130	*
Li-Jaccard [15]	1	1	1	4	176	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Li-NBayes [15]	1	1	1	*	*	*	*	*	*	8	38	62	5	9	9	8	38	62	*	*	*	*	*	*	*	*	*	*	*	*
He-MNBayes [12]	1	1	1	3	3	3	1	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Lu-Levenshtein [17]	1	1	1	1	2	3	1	3	3	*	*	*	1	1	1	*	*	*	3	3	4	1	1	1	4	4	4	*	*	*
Pa-FeaturesSVM [19]	19	68	82	3	45	*	177	*	*	4	21	43	1	10	21	2	14	50	68	*	*	1	3	53	4	4	*	*	*	*
Ca-DLevenshtein [3]	1	1	1	3	19	49	61	*	*	1	1	2	1	2	3	1	3	10	1	3	43	4	4	4	1	1	1	*	*	*
Wa-OSAD [26]	1	1	3	2	4	41	27	*	*	1	1	32	1	1	4	1	10	16	*	*	*	4	4	49	4	4	49	*	*	*
Wa-FLevenshtein [26]	1	20	22	1	3	4	52	76	*	8	28	52	1	6	13	1	8	34	11	*	*	1	1	176	4	4	4	*	*	*

attention to the other side of the coin: proposals of defenses to cover these features in order to defend against such attacks. In this section, we describe the list of WF defenses we have surveyed and simulated. Our simulated defenses operate on packet sequences (packet lengths and timings, but no content) rather than raw packets; they do not directly modify packets during page loading. This allows us to observe if a defense is able to cover a feature modified by a generator. We give our list of defenses, present results on how effectively they covered the features we identified earlier, and follow up by measuring and comparing their bandwidth and time overheads.

A. List of defenses

Packet padding

Packet padding refers to adding garbage data to the end of packets in order to obscure their true lengths. Packet padding schemes have been known in the literature; a large number of different padding schemes were analyzed by Dyer et al. [7] and shown to be ineffective against `Li-NBayes` and `Pa-FeaturesSVM`. Packet padding schemes are meant to obscure the unique packet length feature. We implement two packet padding schemes, described as follows:

1) Maximum padding (PadM). All packet lengths are padded to the MTU.
2) Exponential padding (PadE). All packet lengths are padded upwards to the closest power of 2, but not exceeding the MTU.

No extra packets are added in either of the two schemes. `PadE` is meant to be a bandwidth-cheaper version of `PadM`. The effect of using Tor is similar to `PadM`, as Tor traffic is delivered in fixed-size cells.

Traffic morphing (Wr-Morph)

Wright et al. [27] published a WF defense that is meant to obscure packet lengths, known as traffic morphing. Unique among the defenses, this defense is designed to allow the client to set a target page C_T and mimic the page by modifying packet lengths.

The defense first learns the packet size frequencies of the target page C_T to construct the distribution W_{C_T}:

$$W_{C_T} = \{(\ell_1, f_1), (\ell_2, f_2), ..., (\ell_n, f_n)\}$$

In the above, ℓ_i are packet lengths and f_i are their corresponding normalized packet length frequencies, such that $\sum f_i = 1$. The packet sequence P is morphed so that its packet length distribution is the same as C_T as follows. A matrix M is learned for which each row represents a potential packet length in P and each column represents a packet length in W_{C_T}; the element $M_{i,j}$ of the matrix M is the probability the packet length i would be morphed into packet length ℓ_j. The simplest matrix which satisfies the above is one with each row being $(f_1, f_2, ..., f_n)$; this effectively just samples packet sizes from the target distribution, and ignores the source distribution entirely. Wright et al. minimized the *overhead* (the expected amount of padding added) by solving a quadratic optimization problem over the source *and* target distributions; the optimal solution then yields the desired morphing matrix. This process is done separately for outgoing and incoming packets.

It should be noted however that simple padding of course cannot be used to transform a larger packet into a smaller one. When this is required, Wright et al. suggest that we repeatedly sample directly from the distribution W_{C_T} and ignore the matrix until the (longer) packet has been fully transmitted. We point out that this strategy violates the quadratic optimization. If the client will often morph larger packets into smaller ones, then the resultant traffic will deviate far from the optimized minimal overhead. On the other hand, if the client rarely or never has to morph higher length packets into lower length packets, then the total overhead is the same no matter what the matrix M is (it is roughly equal to the length of P multiplied by the frequency-weighted average packet length in W_{C_T}). These observations combined imply that the optimization stage will either be weakened or insignificant. We have found no easy resolution to this problem, and we will therefore simply sample from the target distribution for all packets, which offers the same defensive properties as any matrix M (but potentially different overhead). In our implementation, we use `google.com` as C_T, our target morph page; it is reasonable for the client to attempt to hide her traffic features by using the most commonly accessed page as a decoy.

HTTP Obfuscation (Lu-HTTPOS2)

HTTPOS (HTTP Obfuscation) was presented by Luo et al. [18] in 2010 as a platform for website fingerprinting defenses. Unlike many other defenses, HTTPOS is implemented entirely client-side, with no need for support from any proxy or the end server. It does so by cleverly using TCP advertised windows, HTTP pipelining, and HTTP ranges in order to control the sizes of both outgoing *and incoming* packets.

HTTPOS contains four modules. The first module is used only when the page is not known to HTTPOS. The third and fourth modules are used to reduce round-trip times and speed up page loading; they do not offer defensive properties. We focus on the second module. In terms of packet sequences, the operations induced by HTTPOS module 2 are relatively simple: Given any incoming packet sized ℓ where ℓ is not the MTU, HTTPOS chooses a uniformly random $0 < r < \ell$, and splits the packet into two packets of size r and $\ell - r$.

The authors also described several other possible defenses, such as manipulating interpacket timing and sending dummy packets, but the implementation of these are not described. We acknowledge that these defenses are possible within their platform; we implement and analyze only the above strategy.

Our simulated implementation is not done at the application layer; rather, we simply go through the packet sequence and split up each incoming packet of size less than the MTU. In the implementation of HTTPOS, this requires a new outgoing packet between the two splits which signals the end server that the client is ready to receive the second split, which costs one round-trip time. In our simulated defense, we assume this can be done without the signal and round-trip time, which is possible with the cooperation of a proxy or the end server.[3] We complete this defense by also padding all outgoing packets to a fixed size of 1500; the authors describe how outgoing packet padding can be done, but they are not clear as to what they implemented. Our choice gives maximum protection for unique packet lengths, at the cost of extra overhead.

Background noise (Pa-Decoy)

Background noise can be used to add randomness to each page load in order to make them less distinguishable. Tor has some background activity that makes fingerprinting more difficult, such as circuit construction, circuit measurement, and stream SENDMEs.

Panchenko et al. proposed a simple defense using background noise to defeat their own attack [19]. Whenever a page is loaded, a decoy page is loaded in the background, using its own connections and connection limit so as not to interfere with the connections of the intended page. This defense has a high overhead. We used a different page from Alexa's top 800 sites as background noise for each of the training and testing elements in order to simulate the intended effect that the attacker cannot predict which page the client picked. Our simulated defense assumes that the decoy page does not interfere at all with the page load of the true page.

[3]This assumption is reasonable for us as many of the other defenses also require the cooperation of a proxy or the end server.

Tor pipelining and request order randomization

Tor developers implemented a prototype defense with no bandwidth overhead in response to the `Pa-FeaturesSVM` attack [20]. The defense is entirely applied at the application layer: HTTP pipelining is enabled, and when a connection becomes idle and a request is selected from the request queue, a random request is chosen rather than the first-in request. The maximum connection limit is also frequently randomized.

As our defense analysis framework is at the network layer, not the application layer, we do not simulate this defense directly. Rather, we note that it acts as a set of transpositions, with some outgoing packets merged together. We can therefore estimate its effect on various attacks from the sensitivity of those attacks to transpositions (G_7, G_8 and G_9 in Table II). Previous authors have shown [3], [26] that a version of this defense is not effective against the latest attacks, such as `Ca-DLevenshtein` and `Wa-OSAD`, as these attacks are sensitive to packet length frequencies as well.

Buffered Fixed-Length Obfuscator (Dy-BuFLO)

After Dyer et al. analyzed a large number of traffic analysis countermeasures and found that efficient defenses failed, they presented their own Buffered Fixed-Length Obfuscator defense and demonstrated its relative success against the attacks of the day [7]. The BuFLO defense is an obfuscator with large bandwidth and time overhead that is applied to both the incoming and outgoing traffic. Packets are sent over a possibly wiretapped connection at fixed intervals with fixed length, and if no data needs to be sent, dummy packets are sent instead. The traffic must continue for a minimum amount of time, after which it may terminate if no more data needs to be sent.

BuFLO is parameterized by three values: d the fixed size of packets (set to 1500), ρ the interpacket timing (set to one packet per 20 milliseconds), and τ the fixed minimum amount of time for which data must be sent (set to 10 seconds). These parameter settings were the strong (and more bandwidth-intensive) choice presented by Dyer et al., which was able to sharply decrease the accuracy of `Pa-FeaturesSVM`. However, this attack was still much better than random guessing against BuFLO; this is because the load time of many web pages exceeded 10 seconds, so BuFLO could not cover their packet length frequencies.

Cai et al. proposed, but did not evaluate, a modification of BuFLO with varying packet rate [3]. They proposed two changes to BuFLO. First, the packet rate would depend on the page being loaded. Second, the total transmission size would be padded. We could not analyze this modification as

Cai et al. did not elaborate on how to select the packet rate, which significantly affects the overhead and privacy properties of the defense. We will present our own modification of BuFLO, and show it is a more effective and efficient defense than the original, in Section VI.

B. Analyzing defenses

We use our comparative methodology to analyze the above defenses. We consider a defense D to be successful in covering the feature modified by a generator $G^{(v)}$ if after applying D to get $D(P)$ and $D(G^{(v)}(P)) = D(P')$, no difference between them can be observed. If a classifier is able to distinguish between P and P', but cannot distinguish between $D(P)$ and $D(P')$, then D seems to be successfully covering the distinguishing feature.

We specify what we mean by "no difference observed" as follows. We use four *feature classifiers*, each one specializing on one specific feature. The defense D is applied to each element in the training set C_{train} and the generator-modified training set C'_{train} to produce $D(C_{train})$ and $D(C'_{train})$; the feature classifiers are trained to distinguish them. D is applied to the testing sets C_{test} and C'_{test} as well, and used to evaluate the feature classifiers. As the objective is simply to find any difference between two classes differing by a single feature, we will specialize our classifiers to focus on one specific feature each. We choose the four feature classifiers as follows:

1) Unique packet length (F_1): Modified `Li-Jaccard`. Given any unique packet length in P_{test}, if it is in any packet sequence $P \in C$ and no $P' \in C'$, add 1 to the score of class C and vice versa.
2) Packet length frequency (F_2): Simplified Naïve Bayes. For training, we simply count the total size sum of incoming and outgoing packets, as well as the total number of incoming and outgoing packets, and take the mean and standard deviation over the packet sequences in each class. Each testing packet sequence is then scored using the normal distribution kernel against those four values for each class, with the incoming and outgoing packets scored separately and then multiplied.
3) Packet ordering (F_3): Half of `Bi-CrossCor`. We use the packet length portion of `Bi-CrossCor`. For a testing packet sequence, each packet length (discarding direction) is compared to the mean of all training packet lengths at that position.
4) Interpacket timing (F_4): Classification based only on total elapsed time. We use this classifier because G_{10} is a delay. This reveals whether or not the total load time of a page would still be a useful feature after the defense is applied.

The results are given in Table III. For each defense, we only present the most successful feature classifier. We only use the interpacket timing classifier on G_{10}. We vary v from 1 to 180 and present the minimum value of v for which the feature classifier had a higher accuracy than 0.55 and 0.75 for all greater v. A lower v indicates that the defense was less successful in covering the feature against our classifiers, whereas an asterisk indicates that our feature classifiers could not expose the feature after the defense is applied. This table also includes our new proposed defense; see Section VI for more details.

We highlight the following results:

1) We can explain prior results from the literature using the table. `PadM`, `PadE` and `Lu-HTTPOS2` have all been shown to be effective against `Li-Jaccard` [15], [18], which is dependent on unique packet lengths; these defenses achieve some success in covering unique packet lengths. `Pa-Decoy` and `Dy-BuFLO` are effective against `Pa-FeaturesSVM` [7], [19], which is dependent on packet length frequencies and these defenses can partly cover packet length frequencies. `PadM` and `PadE` are also shown to be ineffective against `Li-NBayes` and `Pa-FeaturesSVM` [7], as these defenses only cover unique packet lengths but the attacks rely on packet length frequencies. `Ca-DLevenshtein` succeeds against `Lu-HTTPOS2` as it attempts to remove unique packet lengths as a feature, but `Ca-DLevenshtein` is not affected as it is designed for Tor, for which unique packet lengths are not a useful feature. Cai et al. informed the authors of HTTPOS of this result, and the latter modified HTTPOS further, but the modified version is still not effective against `Ca-DLevenshtein` [3].
2) The table exposes several defense flaws. `PadM`, `PadE`, `Wr-Morph` and `Lu-HTTPOS2` are not designed to cover total transmission size (packet length frequencies), so they would be ineffective against the attacks which leverage them. `Pa-Decoy` fails to completely cover interpacket timing because it only covers the total transmission time roughly half the time (when the decoy page takes longer to load than the desired page), which may leak the total page size, a powerful feature. `Dy-BuFLO` does not cover total transmission time if it is over 10 seconds at $\rho = 0.020$ s/packet, which happened quite often. Trying to cover packet length frequency on $G_4^{(v)}$ becomes a race between v and the overhead of `Dy-BuFLO`; a larger v requires a larger setting of minimum time τ to cover it.
3) Tor developers want to understand what WF defenses would work with Tor [21]. As Tor already covers unique packet lengths, `PadM`, `PadE`, `Wr-Morph`, and `Lu-HTTPOS2` are not meaningful on Tor, as all of these defenses are focused on covering unique packet lengths (although only `PadM` truly does so). We note in particular that HTTPOS is a platform valuable for its client-only implementation (requiring no cooperation from a proxy or the end server), but Tor bridges can

Table III

UPPER BOUNDS ON THE QUALITY OF THE DEFENSES. RESULTS ARE GIVEN IN THREE COLUMNS: THE FIRST COLUMN IS THE FEATURE CLASSIFIER THAT WAS ABLE TO ACHIEVE THE HIGHEST MEAN ACCURACY, THE SECOND IS THE VALUE OF v FOR WHICH THE FEATURE CLASSIFIER IN COLUMN 1 HAD AN ACCURACY GREATER THAN 0.55 FOR ALL GREATER VALUES OF v, AND THE THIRD IS SIMILAR, BUT FOR 0.75; ASTERISKS INDICATE WHEN THESE ACCURACIES WERE NOT REACHED FOR $v \leq 180$. ASTERISKS AND HIGHER VALUES INDICATE MORE SUCCESSFUL DEFENSES.

Defense	Unique packet lengths									Packet length frequencies									Packet ordering									Timing		
	G_1			G_2			G_3			G_4			G_5			G_6			G_7			G_8			G_9			G_{10}		
PadM	*	*	*	*	*	*	*	*	*	F_2	16	57	F_2	3	9	F_2	16	57	*	*	*	*	*	*	*	*	*	F_4	23	47
PadE	F_2	28	91	F_3	1	3	F_3	59	*	F_3	2	27	F_2	2	9	F_3	1	1	F_3	29	*	F_3	2	3	F_3	4	4	F_4	23	47
Wr-Morph [27]	F_2	13	74	F_2	29	180	*	*	*	F_2	43	72	F_2	2	11	F_2	32	68	*	*	*	*	*	*	*	*	*	F_4	23	47
Lu-HTTPOS2 [18]	F_3	29	30	F_3	23	50	F_3	45	*	F_3	6	18	F_2	3	9	F_3	1	3	F_3	179	*	F_3	2	*	F_3	4	4	F_4	23	47
Pa-Decoy [19]	F_1	1	68	F_1	48	178	F_1	34	*	F_3	93	*	F_2	38	151	F_3	1	*	*	*	*	*	*	*	F_3	14	*	F_4	111	146
Dy-BuFLO [7]	F_2	147	*	*	*	*	*	*	*	F_2	23	85	F_2	78	*	F_2	20	94	*	*	*	*	*	*	*	*	*	*	*	*
Tamaraw [this work]	F_2	177	*	F_2	180	*	*	*	*	F_2	122	*	F_2	68	168	F_2	82	*	*	*	*	*	*	*	*	*	*	*	*	*

be made to cooperate by implementing a WF defense as a pluggable transport. `Pa-Decoy` and `Dy-BuFLO` achieve only limited success at covering packet length frequencies. In short, none of these defenses can be considered a perfect fit for Tor.

Table IV

SIZE AND TIME OVERHEADS FOR VARIOUS DEFENSE SCHEMES OVER 70 INSTANCES OF ALEXA'S TOP 100.

Defense	Total Size	Total Time
PadM	7.3% ± 0.5%	0% ± 0%
PadE	3.61% ± 0.09%	0% ± 0%
Wr-Morph [27]	29.5% ± 0.8%	0% ± 0%
Lu-HTTPOS2 [18]	8.1% ± 0.2%	0% ± 0%
Pa-Decoy [19]	100% ± 4%	33% ± 6%
Dy-BuFLO [7]	149% ± 6%	310% ± 80%

C. Overhead

We estimate the overhead of each defense in terms of two important metrics: additional bandwidth consumed, and additional time taken. These overhead costs determine the realistic applicability of each defense. On Tor, which is bandwidth starved [22], any bandwidth overhead implicitly reduces the number of clients that will be served. A large time overhead may frustrate users.

We estimate these values by running our simulated defenses on Alexa's top 100 sites. We performed 70 iterations of loading the home page of each of the 100 sites. For `Pa-Decoy`, the decoy page for instance i of page j was set to be instance $i + 1 \mod 70$ of page $j + 1 \mod 100$; the same was done to choose the morph page of `Wr-Morph`. We present the results in Table IV, with means and standard deviations taken over the 70 iterations. We can see that the bandwidth and time overhead for `Dy-BuFLO` are both largest; the defenses which were successful against most feature classifiers, `Pa-Decoy` and `Dy-BuFLO`, have a high total size overhead.

VI. A NEW DEFENSE

In this section we present a prototype of a new defense that can be considered a heavily modified version of `Dy-BuFLO`.

A. Design

Our defense is designed with three goals in mind:

1) Feature coverage. We demonstrate that it is possible to cover all four features in our feature set to a reasonable degree.
2) Overhead reduction. `Dy-BuFLO` is troubled by the dilemma that increasing τ will increase its defensive coverage, but also increase its overhead. We address this dilemma and we are able to find ways to significantly reduce the overhead of `Dy-BuFLO` in both bandwidth and time.
3) Proof of concept. We wish to show that it is not impossible to design a strong defense with lower overhead; there is hope. We are currently working on implementing our defense as a pluggable transport on Tor. As Dyer et al. noted, a perfect implementation of timing intervals is certainly non-trivial [9].

We describe our defense, `Tamaraw`,[4] as follows. As in `Dy-BuFLO`, traffic is still sent in MTU-size packets and at fixed intervals; however, incoming and outgoing traffic are treated differently. Outgoing traffic is fixed at a higher packet interval, which saves overhead as outgoing traffic is much less frequent. We denote the packet intervals as ρ_{out} and ρ_{in} (measured in s/packet). We use experimentation to choose these values in Section VI-B.

`Dy-BuFLO` only attempts to cover total transmission size if the total amount of time for one page is less than τ. This makes the choice of τ especially awkward: increasing τ would increase the number of web pages covered by `Dy-BuFLO`, but it would also increase the overhead. In our new defense, the number of packets sent in both directions are always padded to multiples of a padding parameter, L.[5] This means that if L is large enough, then it is likely that for each web page there exists some other web page that is mapped to the same multiple of L. The way we pad is as

[4] A tamaraw is a lightweight cousin of the buffalo.

[5] It is non-trivial for the proxy to know when to pad, as it does not know when the data stream has ended. One way for the proxy to know this is to set a parameter K, such that if the last K packets were dummy packets, then the traffic is determined to have ended. In our analysis we assume that the client and proxy know when to pad.

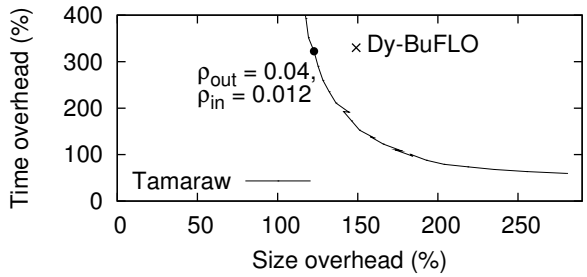

Figure 3. The lower-left boundary of the two-dimensional feasibility region of size and time overhead for Tamaraw when varying ρ_{out} and ρ_{in}. Our chosen parameters and the overhead of Dy−BuFLO on the same data set are marked. The overhead includes the padding mandated by $L = 100$.

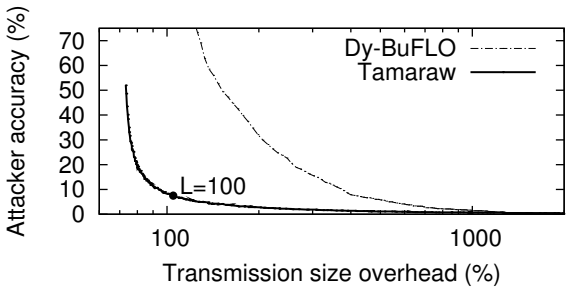

Figure 4. Maximum attacker accuracy against transmission size overhead for Dy−BuFLO (varying τ from 1 to 200 seconds) and Tamaraw (varying L from 5 to 5000) on the same data set. The x-axis is in log scale and starts at 60%.

follows. Suppose the total number of incoming packets is I, where $AL < I \leq (A+1)L$. We pad to $(A+k)L$ at the rate ρ_{in} with probability $(1/2)^k$. This will increase the possible range of I, compared to simply padding to the next multiple of L, which is $(A+1)L$. We do this separately for outgoing packets as well. If the client visits the same page multiple times and the attacker knows this, then this padding scheme effectively reverts to padding to the next multiple of L, as the minimum padding is highly likely to be observed.

B. Experimental results

We experiment with Tamaraw. First, we want to choose ρ_{out} and ρ_{in}. These values were 0.02 for the implementation of Dy−BuFLO, but it is expected that ρ_{out} should not have to be as large as ρ_{in}. As the distinguishability of two different web pages is controlled by the padding parameter L, our objective in the choice of ρ_{in} and ρ_{out} is to minimize overhead. We test the bandwidth and time overhead of Tamaraw on Alexa's top 100 pages, loaded 70 times each. We vary ρ_{out} and ρ_{in} from 0.005 to 0.16 seconds/packet. We present all values for which no other choice of parameters had both a lower time and bandwidth overhead in Figure 3. Generally, as ρ_{in} and ρ_{out} increased, size overhead decreased while time overhead increased. Here we set L to 100, for reasons discussed below. With $\rho_{out} = 0.04$ and $\rho_{in} = 0.012$, we achieve a 17% decrease of total transmission size overhead from Dy−BuFLO's 149 ± 6% to 123 ± 10%, with the time overhead roughly the same, changing from 330 ± 80% to 320 ± 70%. We will experiment with these parameters as they offer a reduction in bandwidth overhead.

In our implementations of Dy−BuFLO and Tamaraw, we pessimistically required that the original logical ordering of the real packets must be maintained. For example, if the defense allowed an outgoing packet to be sent, but the next real packet to be sent is an incoming packet, then a dummy outgoing packet is sent, even if there are other outgoing packets waiting after the incoming packet. This is to guarantee that the causal order is preserved: it could be that the subsequent outgoing packets *depend* on the incoming packet. This rule has a large effect on the bandwidth. A practical implementation could achieve a lower size and time overhead as re-ordering is possible for both defenses when subsequence is not consequence; our simulations are therefore pessimistic on the overhead but necessary to guarantee correctness.

We apply the same defense methodology in Section V-B, and give the results in Table III, presented earlier. We chose $\rho_{out} = 0.04$, $\rho_{in} = 0.012$, $L = 100$. We can see that Tamaraw is much more successful at protecting all features than other defenses, although it cannot achieve a perfect cover of total packet length frequency either. We set $L = 100$ so that Tamaraw would perform better than previous defenses. Looking at the table, we see that Tamaraw is not perfectly successful against generators that significantly change the total transmission size (including the unique packet length generators G_1 and G_2). As Dy−BuFLO sends more dummy outgoing packets than Tamaraw, Dy−BuFLO is more able to cover changes in outgoing transmission size (G_2, G_5), but it is less able to cover changes in incoming transmission size (G_1, G_4, G_6). We next show why Tamaraw is a more effective defense than Dy−BuFLO.

In order to produce an *upper bound* on the attack accuracy of any classifier on Tamaraw, we experiment on Alexa's top 800 sites, one instance each. We define the *ambiguity set* of each page as the set of pages that look the same to the attacker. For Tamaraw, it is calculated as follows. Let D be Tamaraw where L is set to 0 (no dummy packets appended to the end). Suppose the number of incoming packets for defended packet sequence $D(P)$ is $|D(P)_{inc}|$ and the number of outgoing packets is $|D(P)_{out}|$. Two packet sequences $D(P)$ and $D(P')$ are the same under Tamaraw if they satisfy: $\left\lfloor \frac{|D(P)_{inc}|}{L} \right\rfloor = \left\lfloor \frac{|D(P')_{inc}|}{L} \right\rfloor, \left\lfloor \frac{|D(P)_{out}|}{L} \right\rfloor = \left\lfloor \frac{|D(P')_{out}|}{L} \right\rfloor$. This is the case even if the attacker is able to observe those packet sequences multiple times with the knowledge that they belong to two pages. We only need one instance for each of 800 pages because we make the simplifying assumption that each web page will produce the same number of incoming and outgoing packets on each page load; this assumption is advantageous for the attacker as packet length frequency becomes a more identifying feature.

We then consider those two packet sequences to belong to the same ambiguity set. For Dy−BuFLO, we consider two packet sequences to belong to the same ambiguity set if the total transmission size is the same, as total transmission size is the only observable difference.

The number of ambiguity sets is directly linked to the maximum classification accuracy. For an ambiguity set of size s, the attacker can at best achieve an accuracy of $1/s$ on each element in the ambiguity set. Therefore, the maximum accuracy is the total number of ambiguity sets, divided by the total number of packet sequences (800 in our case).

We plot the maximum attacker accuracy against the transmission size overhead in Figure 4. The results show that Tamaraw is significantly better at reducing the accuracy at any given overhead. For reference, at a size overhead of 130%, there are 553 ambiguity sets (maximum accuracy of 69%) in Dy−BuFLO ($\tau = 9$) and 40 ambiguity sets (maximum accuracy of 5%) in Tamaraw ($L = 182$).

VII. Takeaways and conclusion

As we have observed, the relationship between WF attacks and defenses had been unclear, which raised questions about why and how they worked. In this paper, we developed and tested a new feature-based comparative methodology that classifies and qualifies WF attacks and defenses. We observed how attacks work to uncover certain features that characterize a web page, and how defenses attempt to cover them. We saw that newer attacks are sensitive to more features than older attacks. This allows us to explain why some attacks have achieved a higher success rate than others, and to argue how useful our surveyed defenses would be as a countermeasure to these attacks. Our methodology also exposes some flaws of previous defenses.

While previous work has shown that current WF defenses are either ineffective or inefficient, and while our work has explained these results using a systematic methodology, we argue that the situation is not hopeless for web browsing clients who desire privacy. We propose a new prototype defense, Tamaraw, that is able to reduce the overhead of Dy−BuFLO. Using our methodology, we show that Tamaraw provides better protection against website fingerprinting attacks than all previous defenses.

Acknowledgments

We would like to thank Scott E. Coull, Andriy Panchenko, and Kevin P. Dyer for their correspondence with us, which helped us improve the paper.

References

[1] Alexa — The Web Information Company. www.alexa.com.

[2] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Privacy Enhancing Technologies*, pages 1–11. Springer, 2006.

[3] X. Cai, X. Zhang, B. Joshi, and R. Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 605–616, 2012.

[4] P. Chapman and D. Evans. Automated Black-Box Detection of Side-Channel Vulnerabilities in Web Applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 263–274. ACM, 2011.

[5] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 191–206. IEEE, 2010.

[6] H. Cheng and R. Avnur. Traffic Analysis of SSL-Encrypted Web Browsing. http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps.

[7] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 332–346, 2012.

[8] E. W. Felten and M. A. Schneider. Timing Attacks on Web Privacy. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 25–32. ACM, 2000.

[9] X. Fu, B. Graham, R. Bettati, W. Zhao, and D. Xuan. Analytical and Empirical Analysis of Countermeasures to Traffic Analysis Attacks. In *Parallel Processing, 2003. Proceedings. 2003 International Conference on*, pages 483–492. IEEE, 2003.

[10] Y. Gilad and A. Herzberg. Spying in the Dark: TCP and Tor Traffic Analysis. In *Privacy Enhancing Technologies*, pages 100–119. Springer, 2012.

[11] X. Gong, N. Kiyavash, and N. Borisov. Fingerprinting Websites using Remote Traffic Analysis. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 684–686. ACM, 2010.

[12] D. Herrmann, R. Wendolsky, and H. Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 31–42, 2009.

[13] A. Hintz. Fingerprinting Websites Using Traffic Analysis. In *Privacy Enhancing Technologies*, pages 171–178. Springer, 2003.

[14] S. Jana and V. Shmatikov. Memento: Learning secrets from process footprints. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 143–157. IEEE, 2012.

[15] M. Liberatore and B. Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 255–263, 2006.

[16] R. Lowrance and R. Wagner. An Extension of the String-to-String Correction Problem. *Journal of the ACM (JCM)*, 22(2):177–183, 1975.

[17] L. Lu, E.-C. Chang, and M. C. Chan. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *Computer Security–ESORICS 2010*, pages 199–214. Springer, 2010.

[18] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *NDSS*, 2011.

[19] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 10th ACM Workshop on Privacy in the Electronic Society*, pages 103–114, 2011.

[20] M. Perry. Experimental Defense for Website Traffic Fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting, September 2011. Accessed Feb. 2013.

[21] M. Perry, E. Clark, and S. Murdoch. The Design and Implementation of the Tor Browser [DRAFT]. https://www.torproject.org/projects/torbrowser/design/. Accessed Oct. 2013.

[22] R. Pries, W. Yu, S. Graham, and X. Fu. On Performance Bottleneck of Anonymous Communication Networks. In *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–11, 2008.

[23] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *2002 IEEE Symposium on Security and Privacy*, pages 19–30. IEEE, 2002.

[24] Tor. Tor Metrics Portal. https://metrics.torproject.org/. Accessed Oct. 2013.

[25] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition*. Nauka, 1974.

[26] T. Wang and I. Goldberg. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, 2013.

[27] C. Wright, S. Coull, and F. Monrose. Traffic Morphing: An Efficient Defense against Statistical Traffic Analysis. In *Proceedings of the 16th Network and Distributed Security Symposium*, pages 237–250, 2009.