Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks

Tao Wang
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
taow@cse.ust.hk

Ian Goldberg
Cheriton School of Computer Science
University of Waterloo
iang@cs.uwaterloo.ca

Abstract

Website fingerprinting (WF) is a traffic analysis attack that allows an eavesdropper to determine the web activity of a client, even if the client is using privacy technologies such as proxies, VPNs, or Tor. Recent work has highlighted the threat of website fingerprinting to privacy-sensitive web users. Many previously designed defenses against website fingerprinting have been broken by newer attacks that use better classifiers. The remaining effective defenses are inefficient: they hamper user experience and burden the server with large overheads.

In this work we propose Walkie-Talkie, an effective and efficient WF defense. Walkie-Talkie modifies the browser to communicate in half-duplex mode rather than the usual full-duplex mode; half-duplex mode produces easily moldable burst sequences to leak less information to the adversary, at little additional overhead. Designed for the open-world scenario, Walkie-Talkie molds burst sequences so that sensitive and non-sensitive pages look the same. Experimentally, we show that Walkie-Talkie can defeat all known WF attacks with a bandwidth overhead of 31% and a time overhead of 34%, which is far more efficient than all effective WF defenses (often exceeding 100% for both types of overhead). In fact, we show that Walkie-Talkie cannot be defeated by any website fingerprinting attack, even hypothetical advanced attacks that use site link information, page visit rates, and intercell timing.

1 Introduction

Website fingerprinting (WF) attacks are classification attacks that allow a local, passively observing eavesdropper[1] to determine which web page a client is visiting by observing the client's sequence of packets. WF attacks succeed against clients using privacy technologies, such as VPNs, IPsec, and Tor, that hide the contents and destinations of packets. The attacker—such as the client's ISP or government—uses various packet sequence features, such as packet counts, packet order, packet directions, and unique packet lengths to classify the web page [5]. WF attacks require only local eavesdropping capabilities, small computational cost, and carry little risk of detection. As web-browsing clients of these privacy technologies do not want to reveal the web pages they are visiting to any eavesdropper, they need to *defend* their privacy against WF in some way.

Website fingerprinting is a well-established threat to privacy in the literature [8, 14, 23], as well as in practice: Tor, a popular anonymity network, has implemented a WF defense [24, 26]. However, Tor's defense does not succeed in lowering the accuracy of WF attacks [6, 31]. Researchers have proposed alternative defenses, but these defenses are either ineffective against newer attacks [31] or carry a very large overhead [4, 8, 20, 31]. We describe previous website fingerprinting work in detail in Section 2.

In this paper, we present **Walkie-Talkie (WT)**, a new WF defense, with the following properties:

1. Effective: Many WF defenses have failed against newer WF attacks. WT succeeds against all known WF attacks, including attacks that leverage timing and packet ordering.

2. Efficient: A high bandwidth overhead burdens the network, while a high time overhead frustrates the user. (We define these terms rigorously in Section 3.2.) WT requires a much smaller overhead than all known effective defenses.

3. Easy to use: WT requires no changes to web servers and therefore does not impact server performance, as it needs to be deployed only on the client and proxies. Our implementation only modifies the application layer. Furthermore, the defense can be de-

[1] Active eavesdroppers are not considered WF attackers in the literature, as later explained in Section 2.

ployed incrementally as it does not depend on other clients using the same defense.

Walkie-Talkie consists of two components: half-duplex communication and burst molding. We describe both components in Section 4. These components transform packet sequences of monitored sensitive pages and benign non-sensitive pages, so that these packet sequences are exactly the same (each packet has the same timing, length, direction and ordering). Since the packet sequences are exactly the same, and WF attacks are based solely on classifying packet sequences, no WF attack can succeed against Walkie-Talkie. To mold sensitive packet sequences into non-sensitive packet sequences, the client would need to have some information about them. We will show that such information can be practically obtained and delivered to the client.

For the purposes of this paper, we base our experiments and implementation on Tor, though Walkie-Talkie works on any other setting where website fingerprinting is a threat (using encryption with proxies to hide from a local attacker). We evaluate Walkie-Talkie on a data set collected over Tor, squaring off our defense against known attacks and other known defenses in Section 5. We show that known website fingerprinting attacks are unable to succeed against packet sequences under Walkie-Talkie, and that our defense has a significantly lower overhead compared to known defenses. We describe ways to defeat a hypothetical attacker using more advanced strategies beyond known website fingerprinting attacks in Section 6. We conclude in Section 7, and we include a link to share our code and data in the Appendix.

2 Related Work

Remote side-channel analysis can be used to attack web clients in a wide range of scenarios, including network timing attacks [3], cache attacks [21], and browser fingerprinting [9]. Some of these involve an active attacker, for example one that may send JavaScript requests when the client visits an attacker-controlled web page. This work focuses on defeating website fingerprinting (WF), where the attacker is passively monitoring web packets. Researchers have identified WF as a potential attack against privacy since 1998 [7]. WF has become especially relevant with the growing popularity and usability of privacy technologies such as Tor and the revelation that state-level adversaries are willing to eavesdrop on Internet users en masse [11]. As a result, Tor currently employs a WF defense [24]. In this section, we discuss known WF attacks and defenses to contextualize our work.

2.1 Attacks

There is a long line of research on WF attacks [6, 12, 13, 15, 16, 22, 23, 30, 31]. In WF, the attacker classifies which web page each testing packet sequence belongs to. To do so, the attacker learns to classify using a set of training packet sequences and a machine learning technique. In the *closed-world scenario*, testing packet sequences come from a (small) list of monitored sensitive web pages the attacker knows, and the attacker must distinguish packet sequences coming from each of those pages. In the more realistic *open-world scenario*, testing packet sequences could also originate from non-sensitive web pages outside of the list and unknown to the attacker. In the open-world scenario, the attacker needs to distinguish between sensitive web pages and be able to identify that a non-sensitive web page is non-sensitive.

Over time, researchers have demonstrated increasingly accurate [22] and noise-tolerant attacks [33] using better classifiers. While older attacks were only able to identify pages in the closed-world scenario, newer attacks are also able to tackle the open-world scenario, thus posing a practical threat to privacy. We refer the reader to previous work [5, 22, 31] for a more detailed discussion of the specific workings of each WF attack and how they have evolved.

2.2 Defenses

Wright et al. (2009) published traffic morphing [34], a defense that randomly pads unique packet lengths so that these packet lengths look as if they came from another distribution of packet lengths corresponding to another web page. They showed that this defense was effective against an earlier attack (2006) by Liberatore and Levine [15], because that attack relies on unique packet lengths and does not consider other features such as packet ordering. Later, Wang et al. (2014) showed that this defense was not effective against their new attack, which uses packet ordering as a feature [31].

Luo et al. (2011) published HTTPOS (HTTP Obfuscation) [17]. They implemented the defense on the client side using features in HTTP: the client sets a `Range` header in order to split traffic into packets of random length and uses HTTP pipelining to change the number of outgoing packets. Luo et al. have shown that this is a successful defense against older attacks [2, 15, 30], but other researchers have also found that it is not a successful defense against several newer attacks [6, 31].

Tor has implemented another WF defense [24] in response to a WF attack by Panchenko et al. [23]. Tor's defense uses HTTP pipelining by randomizing the maximum number of requests in a pipeline, so that the order of requests may change if the number of requests exceeds the depth of the pipeline. This defense has no bandwidth overhead as pipelining does not introduce extra packets. Tor has updated its defense [26] recently in response to newer attacks, but both versions of Tor's defense have little effect on the accuracy of known attacks [6,31,32].

We are aware of six WF defenses that are still effective: Decoy (Panchenko et al. 2011 [23]), BuFLO (Dyer et al. 2012 [8]), Tamaraw (Cai et al. 2014 [5]), CS-BuFLO (Cai et al. 2014 [4]), Supersequence (Wang et al. 2014 [31]), and Glove (Nithyanand et al. 2014 [20]). We refer to BuFLO, Cs-BuFLO, and Tamaraw as BuFLO defenses, as the latter two are modifications of BuFLO to lower overhead. Supersequence and Glove share the same usability issue as our work: they require the client to have some information about web pages. Whereas the issue is a stumbling block for Supersequence and Glove, our work resolves this issue by using half-duplex communication to minimize the amount of information the client needs to have, which we describe in detail in our evaluation (Section 5). All of these previous effective defenses generally require more than 100% bandwidth and/or time overhead.

3 Preliminaries

3.1 Attack Scenario

We consider a web-browsing client that is connecting to the Internet using one or more proxies over an encrypted connection. A packet received over such a network (e.g., a TLS packet) at some time t and having some length ℓ is denoted as $p = (t, \ell)$. A packet sequence is denoted as $s = \langle p_1, p_2, \ldots \rangle$. We use positive lengths to denote outgoing packets from the client and negative lengths to denote incoming packets.

We assume the attacker is *local* to the client and *passive*, consistent with previous works on website fingerprinting. Possible local attackers may include the client's ISP, wiretappers, packet sniffers, and other eavesdroppers. Since the attacker is local, the attacker knows the client's identity, but does not know which page she is visiting because she is using one or more proxies. As a passive eavesdropper, the attacker never attempts to modify the client's packet sequence. The attacker is therefore very hard to detect.

The attacker seeks to identify static web pages; Walkie-Talkie does not protect dynamic content. It is difficult to defend dynamic content as a whole, as bandwidth and timing requirements vary significantly. For example, it would be overwhelmingly expensive to make an online chatroom confusable with a high-quality video stream. Some types of dynamic content are not susceptible to WF, such as chatting and file downloading. Other works have shown that search queries [18] and videos [27] are susceptible to fingerprinting attacks. As pages are static, they are associated with finite-length packet sequences.

In our scenario, at least one proxy is outside the WF attacker's control. Otherwise, the attacker has already won without the need of website fingerprinting: previous work has shown that an attacker with control over both ends of a multi-proxied connection can compromise the client's privacy completely [19]. The non-compromised proxy (which we simply refer to as the proxy hereafter) is willing to protect the client's privacy by shaping the traffic according to her specification. A proxy who shapes the traffic incorrectly can be easily detected by the client, who sees the whole packet sequence.

As a preliminary, the client and proxy implement a simple defense: all packets they send to each other are of the same length, much like in Tor. They can do so by splitting longer packets and padding shorter ones. Previous work has shown that TCP packet lengths leak too much information to the WF attacker [5]. Indeed, Tor relays use fixed-length *cells* to deliver information; for this reason, previous work has found that Tor is much harder to attack with WF than many other web privacy technologies [13], though Tor is still vulnerable. Borrowing Tor's terminology, we use the term "cells" instead of "packets" to describe the fixed-length data elements, and scale our size units so that a cell has $|\ell| = 1$. Note that although we borrow the fixed-size cell concept from Tor, our defense is nevertheless applicable to other technologies such as VPNs and IPsec.

3.2 Overhead

To show that WT is efficient, we will evaluate its bandwidth overhead and time overhead.

The **bandwidth overhead** of a defense is the number of dummy cells added by the defense, divided by the number of cells in the undefended (original) cell sequence. Dummy cells are necessary to obfuscate the true amount of data on the wire. Bandwidth overhead represents a burden to the proxy and possibly other proxies between the client and the proxy. Note that the web server does not suffer from bandwidth overhead; it will never generate or see dummy cells.

The **time overhead** of a defense is the extra amount of time required to load the cell sequence, divided by the original amount of time required. To keep bandwidth overhead and time overhead separate, we assume that dummy cells do not add to the time overhead by themselves (i.e., the bandwidth is sufficient that extra dummy cells can be sent without delaying real cells). Nevertheless, all known effective WT defenses incur a large time overhead, typically because they artificially delay cells in order to induce desired traffic patterns such as sending cells at a constant rate. A large time overhead deteriorates the client's experience, as the client needs to wait longer to load web pages, but it does not burden the proxies.

4 Components of Walkie-Talkie

Walkie-Talkie consists of two components: *half-duplex communication* and *burst molding*. To defend a cell sequence from a sensitive web page, half-duplex communication transforms the cell sequence into a burst sequence. which is then molded into a burst sequence from a non-sensitive web page. We describe both components and how they work together in detail in this section.

4.1 Half-Duplex Communication

We modify the client's web browser so that it communicates in half-duplex mode, much like a walkie-talkie. Normally, web browsing is full-duplex: multiple servers are sending web page data to the client while the client simultaneously sends further resource requests, possibly to new servers. The pattern of exactly when the client has received, for example, an `img` tag within an HTML resource, causing it to immediately fetch the corresponding image resource, is a strong feature for the WF attacker. Under our defense, the client only sends requests after the web servers have satisfied all previous requests. As a result, the client and proxy both send data in interleaving bursts of incoming and outgoing cells. Walkie-Talkie does not affect web servers.

The goal of half-duplex communication is to reduce the information available to the WF attacker about the cell sequence s to the form $s = \langle (b_{1+}, b_{1-}), (b_{2+}, b_{2-}), \ldots \rangle$, a *burst sequence*: each b_{i+} is the number of continuous outgoing cells sent in a burst and each b_{i-} is that for the succeeding incoming cells. We can think of half-duplex communication as a way to group same-direction cells together.

The benefit of using burst sequences instead of cell sequences is that they can be *molded* at little overhead, and molding them is computationally cheap. (We describe molding in detail in Section 4.3). Indeed, previous defenses (Supersequence [31], Glove [20]) have attempted to mold cell sequences directly, at a much greater cost in overhead. Another issue with these previous defenses is that they require the client to know the cell sequences of many pages, but cell sequences carry a lot of information and are therefore difficult to deliver and store. Burst sequences are much lighter in information content, and we will show that it is practical to deliver and store hundreds of thousands of burst sequences.

4.1.1 How browsers work

In this section we describe how browsers use persistent connections to load data from a web server. We use the terminology defined in RFC 7230 on "HTTP/1.1 Message Syntax and Routing", especially its discussion on connection management in Section 6 [10]. While our implementation is based on Tor Browser, any browser with persistent connections (i.e., any browser supporting HTTP/1.1) can be modified to support half-duplex communication.

During web browsing, clients make requests to obtain data from the server (or post data to the server). To send requests, the browser creates or re-uses persistent TCP/IP connections (up to a preset maximum number of connections). When requests are complete, the browser may close the attached connections, or keep them alive as open connections in order to send further requests to the same server.

As the total number of simultaneously open connections is (tightly) limited, a browser will often be unable to make further requests until current requests are completed. Until then, the browser stores the request in a pending request queue. When a request completes or when a connection dies, the browser enumerates the pending request queue in an attempt to send requests (sometimes by creating new connections). During the enumeration process, the browser may re-use open connections or close them to make room for new connections to other servers.

4.1.2 Implementation of half-duplex mode

We add two states to the browser to enforce half-duplex communication: *walkie* and *talkie*. Conceptually, the *walkie* state corresponds to an idle browser; the *talkie* state corresponds to a browser that is actively loading a page (which may be any number of resources). We explain each below. Our modification only adds 26 lines of code and removes 12 lines of code from the connection manager in Tor Browser (which is itself a modification of Firefox), and it is available for download with a link in Section 7.

The browser starts in the *walkie* state. When the client starts any request while in the *walkie* state, the browser sends the request immediately, and the browser switches to the *talkie* state. After the page has finished loading, when there are no pending requests left, the browser will return to the *walkie* state.

In the *talkie* state, the browser is currently loading a page. The browser always queues new requests in this state; it never sends requests immediately. Furthermore, the browser does not enumerate the pending request queue whenever any connection dies or become idle. Rather, the browser only enumerates the request queue and sends out requests when there are *no active connections left* (i.e., all connections have died or become idle). If instead the request queue is empty, the browser returns to the *walkie* state; page loading has stopped.

We justify why the above states implement half-duplex communication by making the following observation: the client never attempts to initiate new HTTP requests when there are any active connections left. This is true in both the *walkie* and the *talkie* state. Since an HTTP server does not actively initiate contact with the client, the lack of active connections means that the server is never sending data when the client initiates new HTTP requests.

However, the above alone is not sufficient to ensure half-duplex communication. This is because making a new HTTP request is not instantaneous. Unless a pre-existing open connection to the server exists, the client must spend an extra round-trip time to open a new connection. The round-trip time creates a time gap that causes the client to talk when the servers are already responding to other HTTP requests. One way to solve this problem is to ensure that the client must establish a connection and send the HTTP request in two bursts rather than one burst. We implement a more efficient solution to this problem, as described below.

4.2 Optimistic data

Normally, when a client wishes to load a resource from a web server, the client makes a TCP connection request, waits for the server's request acknowledged message, and only then will the client send a `GET` request to load the resource. This creates an extra round-trip time that can be removed by having the client send both the TCP connection request[2] and the HTTP `GET` request at the same time. The final hop holds the `GET` request until the TCP connection is established, and then sends out the `GET` request. This is known as *optimistic data* in Tor, and Tor Browser has used optimistic data since 2013 [25]. As optimistic data works on Firefox in general if the client is using a SOCKS proxy, users of other privacy options and anonymity networks can use optimistic data as well.

Optimistic data works at the socket level. Normally, after sending a connection establishment request, the socket waits for an acknowledgement by the server before informing the browser that it is ready to send requests. With optimistic data, the socket does not wait, but rather it immediately pretends to the browser that the server has established the TCP connection, which causes the browser to send the `GET` request immediately. Optimistic data is useful for our defense, as it allows the client to establish a new connection and send the relevant request at the same time. Optimistic data reduces the number of bursts and thus the amount of padding we need to confuse the attacker.

[2]The TCP connection request is here an *application-layer* message instructing the last hop in the anonymity network to make a TCP connection to the desired destination.

4.3 Burst molding

Burst molding draws from the concept of Decoy, the WF defense described by Panchenko et al. [23], which loads two pages in parallel to confuse the adversary, at an approximately 100% bandwidth overhead. The adversary cannot determine which of the two pages is really visited by the client. We can further leverage the open-world scenario to improve the defense mechanism: if the real page is a non-sensitive page, we will choose a sensitive page as the decoy page, and vice versa. If the client's sensitive pages are always loaded with popular non-sensitive pages, the attacker can never determine that she has visited a sensitive page. This is especially effective if the non-sensitive page is sufficiently popular, in which case the attacker suffers from the base rate fallacy. It is plain to see that Decoy is effective no matter what classifier the WF attacker uses. Burst molding is able to achieve the same property.

However, instead of actually loading two pages, burst molding *simulates* loading two pages by loading the *supersequence* of two burst sequences, which allows a much lower overhead than loading two pages. A sequence s' is a supersequence of s if s' contains s; this applies to both cell sequences and burst sequences. The idea of simulating supersequences is inspired by Supersequence [31] and Glove [20]. Allegorically, adding padding cells is like injection molding: burst molding adds cells to the original burst sequence so that it is molded into the supersequence.

Burst molding adds fake cells to burst sequences as follows. If the number of cells in a burst of the real page is $b_i = (b_{i+}, b_{i-})$, and for a burst of the decoy page it is $b'_i = (b'_{i+}, b'_{i-})$, we will send $\hat{b_i} = (\max\{b_{i+}, b'_{i+}\}, \max\{b_{i-}, b'_{i-}\})$ cells. We do so for every burst in each burst sequence. If the number of bursts in the two burst sequences is different, we add fake bursts consisting of entirely fake cells to the shorter sequence. We do so for each burst, resulting in a significantly lower overhead compared to simply loading two pages at once: burst molding uses the *max*, while Decoy would use the *sum* of burst sequences. The attacker knows that any subsequence of the above is possibly the real page—including the real and decoy pages themselves—but cannot tell which is the real page.

Fake cells in a burst add to the bandwidth overhead, but do not add to the time overhead (according to our definition in Section 3.2). Fake bursts consisting of en-

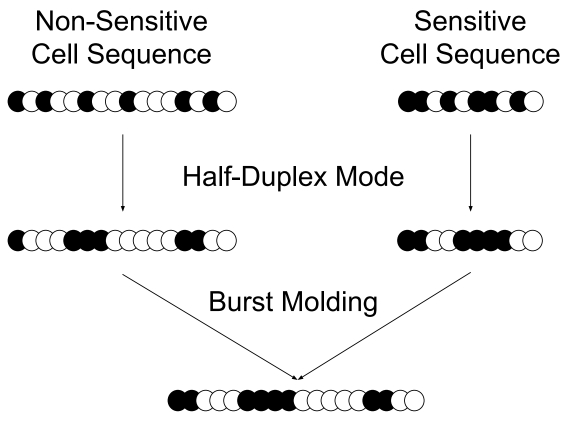

Figure 1: Diagram showing the effect of Walkie-Talkie on cell sequences. Black circles indicate outgoing cells and white circles indicate incoming cells. Walkie-Talkie consists of two steps: half-duplex mode and burst molding. Half-duplex mode groups cells of the same direction together, while burst molding adds fake cells to make sensitive and non-sensitive cell sequences the same.

tirely fake cells add to both the bandwidth and time overhead. We show the effect of half-duplex communication and burst molding in Figure 1 as an illustration.

4.3.1 Advantages

We will show that burst molding is more effective and has lower overhead compared to other defenses in Section 5. Burst molding has several other qualitative advantages, which we describe below:

Cover story.
With burst molding, the client knows and could control what non-sensitive web pages have been used to disguise her page accesses. This gives the client an explicit cover story for her actions. This is not the case in BuFLO [8], Tamaraw [5], and CS-BuFLO [4], where the client cannot know or control which other page her cell sequence appears to come from (rather, the client is only given the assurance that such a page is likely to exist).

Base rate.
Web pages are accessed with vastly different base rates in the real world, but most work in the field (including all defenses) has ignored this fact. Our design specifically takes this into account, as we use more popular (and less sensitive) Alexa's top pages as decoy pages. In the above scenario, an attacker trying to claim the client visited the sensitive page is highly likely to be wrong. We further develop on how varying base rates affect our defense effectiveness in Section 6.2.

Minimizing computation.
Wang et al. pointed out that Supersequence requires the solution of an NP-hard problem [31] to minimize bandwidth overhead for cell sequences. Both Supersequence and Glove use an approximation algorithm to this problem. This approximation algorithm is nevertheless slow, and the client would have trouble computing the supersequence of a large number of cell sequences. For WT, computation of burst supersequences is cheap: we simply take the maximums of several pairs of numbers.

Minimizing client information.
WT, Supersequence, and Glove all require the client to know some decoy pages. The difference is that WT requires burst sequences, whereas the latter two require cell sequences. Burst sequences are much easier to store than cell sequences, because we do not need to store the ordering of cells. On our data, we found that we only need about 20 bytes of information to describe the burst sequence of a web page, whereas cell sequences require 36 kB of information on average; burst sequences are about 1800 times more efficient to store and deliver. For example, the client can know the burst sequences of 100,000 potential decoy web pages by loading and storing 2 MB of data. Currently, a Tor client needs to load about 8 MB of data when starting up Tor for relay discovery, so this amount is feasible on Tor. Tor directory authorities can collaborate with each other to collect cell sequence data, and send the data to clients along with relay data.

4.3.2 Choosing decoy pages

We can optimize the overhead of burst molding by choosing decoy pages cleverly, instead of simply choosing a random burst sequence. For each sensitive page s in our set of known burst sequences, we pre-compute its overhead when sent with each of the set of non-sensitive pages in our set; suppose non-sensitive page s' caused the minimum overhead when sent with s (conceptually, s and s' are similar cell sequences). Then we pair s and s' together, such that when the client needs to visit s, she uses s' as a decoy page; similarly when the client needs to visit s', she uses s as a decoy page. Each decoy page is only paired with one other page. The choice of decoy pages is then symmetric between sensitive and non-sensitive pages, and reveals no information as to which one triggered the cell sequence. This optimization is only possible if the client knows the burst sequence of her real page. In case she does not, she defaults to simply choosing a decoy page randomly. Burst molding is therefore most efficient when the set of decoy pages is large, and we have seen that a large set of decoy pages is practical.

Some clients may not want to use sensitive pages as decoys, as they would rather not attract the attention of eavesdroppers monitoring sensitive page access. It is however necessary that sensitive pages should be used as decoys; otherwise, whenever the attacker detects that the client is visiting either a sensitive page or a non-sensitive page, the attacker would know that the non-sensitive page is a decoy. Further, we argue that the use of a sensitive decoy page is no more compromising than the use of proxies or encryption: for instance, the presence of ciphertext does not suggest that the plaintext is noteworthy. In particular, the client is never made to visit sensitive pages under WT, which is an advantage over the defense of Panchenko et al. [23] She only adds fake cells in a way that matches the burst sequences of sensitive pages.

We evaluate a fixed set of decoy pages in this work, though it is possible for the client to choose her own decoy pages. For example, a German-speaking client may choose popular German pages to be more convincing.

4.4 Practical implementation

In WT, the client and proxy construct the supersequence together by respectively adding fake cells and bursts to their outgoing packets. The client chooses the decoy page and sends the decoy burst sequence to the proxy before starting a page visit. The proxy counts the number of sent packets in each burst and adds packets if it is lower than the number of required packets in the decoy burst sequence. Therefore, there is almost no computation overhead to the proxy. WT is deployable: any proxy that is willing to carry and encrypt traffic for the client would also be willing to mold it slightly for her privacy.

As a proof of concept, we implemented burst molding by modifying the Tor client. Our implementation adds 143 lines of code to Tor. We added two new cell types, a fake cell and a fake burst end cell. During a real burst, the client sends fake cells before sending real cells. The proxy sees the client's fake cells, drops them, and similarly starts sending fake cells before sending real cells. During a fake burst, the client and the proxy both use fake burst end cells to mark the end of each fake burst.

The chief difficulty in our implementation was that the Tor client had to stop delivering cells in the middle of fake bursts. Otherwise, the fake burst would look different from real bursts. We did so by adding a queue to each Tor circuit, so that each cell that was created in between fake bursts would be queued. At the end of a fake burst (signalled by the fake burst end cell), the client empties the queue and sends the queued Tor cells as the next burst. Our implementation does not rely on any Tor-specific mechanics, and could be applied to other proxy technologies.

Our implementation assumes that that the client can either collect burst sequences or receive them from somewhere else. (Our security analysis assumes that the attacker is allowed to see them.) We describe an alternative construction of Walkie-Talkie for which the client has no information about any real burst sequences in Appendix A.

4.5 Security

We analyze the security of Walkie-Talkie against an attacker who wants to know when a client has visited some sensitive page s. The client really visits s at probability p and chooses s as a decoy page for some other page with probability p'. It is plain to see that the attacker's precision cannot exceed $p/(p+(1-p)p')$, as no attacker can distinguish between real visits and decoy visits.

To achieve the maximum precision, the attacker must be able to correctly determine the two subsequences that make up any given cell sequence of WT. We will see in the evaluation (Section 5.2) that no real attack comes close to doing so. Even a theoretical perfect classifier fails to do so; in Section 5.4.2, we show that there are often hundreds if not thousands of possible realistic subsequences (from a set of 10,000 subsequences) for any given cell sequence of WT.

We extend our analysis to include the scenario that the client may not have chosen s as a decoy page. Consider two types of clients: clients who really visit s at some probability p taken from distribution X, and clients who only use s as a decoy with probability p' taken from some distribution X'. To distinguish between those two types of clients, the attacker must be able to judge if the client's visits of s come from X or X'. It is not practical for the attacker to do so, as the attacker cannot directly measure X, X changes over time in an unpredictable manner, many page visits would include s as a subsequence (even without choosing s as either a real page or a decoy page), and the attacker's estimation of X' is significantly affected by observation error, especially if the set of decoy pages is rotated regularly. Therefore, the attacker cannot determine if any given client has ever really visited s, or merely uses it as a decoy page.

5 Evaluation

Here we evaluate WT on data collected from Tor using the methodology described next in Section 5.1. In Section 5.2 we show that WT is effective against known WF attacks. In Section 5.3 we compare our defense against known defenses to show the significantly lower overhead of WT. WT is in fact effective against all possible WF attacks; we rigorously define this notion and quantify WT's effectiveness against all WF attacks in Section 5.4.

5.1 Setup and Data Collection

We collected our data on Tor Browser 6.0 (based on Firefox 38.7.1) with Tor 0.2.8.1. To collect burst sequences for WT, we modified Tor Browser to enable half-duplex communication, as described in Section 4.1.2.

We collected data from Alexa's top pages [1]; we use long-standing pages to make our results more reproducible and comparable to other papers in the field. We use 100 of the top pages as the non-sensitive set (after removing duplicates due to different localizations or URLs of the same page), and we collected 100 instances of each page in the non-sensitive set. We use the next 10,000 pages in Alexa's top pages as the sensitive set. In the closed-world scenario, we only use the former data set, in which case the 100 top pages are sensitive instead; to avoid confusion, in this case we refer to the top 100 pages as the closed-world set. We dropped any instance with fewer than 50 cells (25 kB) in it, in order to discard pages that failed to load.

We added the capability to generate fake cells on Tor clients and relays, but we will not use the latter to achieve burst molding in this section. Rather, we will simulate burst molding after collecting data using half-duplex mode. This is because we want to present experimental results for a large number of parameter choices for burst molding, and re-collecting data for each set of parameters is infeasible. Our simulated burst molding does not consider network instability events such as packet loss and proxy dropping; these events are rare and unlikely to be caused by and therefore linkable to the server.

5.2 Walkie-Talkie versus Attacks

We implemented nine known WF attacks and tested each of them against WT. Each WF attack we tested was the state of the art at the time of its publication. Since many of the older attacks were not designed for the open-world scenario, we tested all of them in the closed-world scenario for consistent comparison. We use 100 instances of each of the 100 closed-world pages for training and testing with 10-fold cross validation. Since the closed-world scenario is strictly easier to attack than the open-world scenario, our results are a conservative estimate of WT's effectiveness.

We show the results in Table 1 under two columns: the original accuracy on a Tor data set without our defense (Undefended), and the new accuracy on a Tor data set with our defense (Defended).

Jaccard and MNBayes are highly inaccurate even in our Undefended case because they rely on unique packet lengths, but all of our cells have the same length (see Section 3.1). Out of all the attacks, SVM by Panchenko et al. [23] appears to suffer least from WT, performing slightly better than kNN [31]. Indeed, previous authors [5,8] have noted the resilience of this attack against random noise, possibly due to its use of a "kernel trick" transforming distances between cell sequences, allowing greater flexibility in ignoring dummy cells. While our experiments on the closed-world scenario show that WT is successful, WT truly shines in the more realistic open-world scenario, which we investigate next.

We designed WT for the open world, as it attempts to confuse sensitive and non-sensitive pages. We focus on three WF attacks that have been successful in the open-world scenario: SVM, kNN, and CUMUL, and present their TPR and FPR in Table 2. We see that the FPR for each attack increases significantly with the application of WT. kNN adopts an aggressive strategy, achieving a high TPR but suffering a high FPR, whereas CUMUL and SVM both suffer a low TPR with a low FPR.

The base rate fallacy tells us that since the TPR and FPR are similar for all three attacks, they are highly imprecise if the base rate of sensitive page access is low. This is an important consideration as realistically, clients do not often visit sensitive pages. For example, if the rate

Table 1: Closed-world accuracy (TPR) of known attacks against Tor (Undefended), and Tor protected by WT (Defended).

Attack	Undefended	Defended
Jaccard [15]	0.01	0.01
Naive Bayes [15]	0.49	0.16
MNBayes [13]	0.03	0.02
SVM [23]	0.81	0.44
DLevenshtein [6]	0.94	0.19
OSAD [32]	0.97	0.25
FLevenshtein [32]	0.79	0.24
kNN [31]	0.95	0.28
CUMUL [22]	0.64	0.20
kFP [12]	0.86	0.41

Table 2: Open-world accuracy (TPR and FPR) of known attacks against Tor (Undefended), and Tor protected by WT (Defended).

	True Positive Rate (TPR)	
Attack	Undefended	Defended
SVM [23]	0.47	0.33
kNN [31]	0.98	0.68
CUMUL [22]	0.78	0.20

	False Positive Rate (FPR)	
Attack	Undefended	Defended
SVM [23]	0.05	0.20
kNN [31]	0.09	0.62
CUMUL [22]	0.04	0.35

Table 3: Accuracy of each feature category of kNN against Tor (Undefended), and Tor protected by WT (Defended).

Category	Undefended	Defended
Sequence length	0.67	0.14
Location of outgoing cells	0.01	0.01
Ratio of outgoing cells	0.79	0.19
Cell bursts	0.81	0.27
Direction of initial cells	0.04	0.01
Intercell times	0.10	0.04

of sensitive page access is 5%, then kNN would have a precision of only 5.5%; almost all of its sensitive classifications are wrong. Despite having a decent recall rate, kNN would be useless against WT as the attacker cannot act upon its sensitive classifications.

We seek to delve deeper into the success of WT against known WF attacks by examining how WT affects individual features. To do so, we examine the feature categories defined by kNN [31]. We choose kNN because its feature categories are diverse and understandable, and it is one of the better attacks. Returning to the closed-world scenario for this experiment, we measure the effectiveness of each individual category by calculating the classification accuracy if only features from that category were used for kNN classification. We contrast the effectiveness of each category before and after WT is applied on our cell sequences.

We plot the six feature categories and their results in Table 3. Each feature category that was useful for classification in the Undefended case has been covered by WT. Although WT makes no explicit attempt to cover intercell times, the addition of fake cells appears to disrupt intercell times as a feature. Comparing Table 3 and the entry for kNN in Table 1, we see that the accuracy of kNN under WT would be almost unchanged if only the sizes of the cell bursts were used and other feature categories were discarded. This reflects the fact that WT effectively reduces the information available to the attacker to simply the burst sequences.

5.3 Walkie-Talkie versus Defenses

In the other direction, we compare WT with a basket of known website fingerprinting defenses in Table 4, in terms of bandwidth overhead (BWOH), time overhead (TOH), and accuracy of the kNN attack by Wang et al. [31]. We use the kNN attack because it is the current state-of-the-art attack on Tor. We implemented all of these attacks based on their original authors' descriptions. We did not include some older defenses which had no effect on cell sequences, as they only affected packet sizes.

Table 4: Bandwidth overhead (BWOH) and time overhead (TOH) of the best WF defenses, as well as the accuracy of kNN on them in our data set.

Defense	BWOH	TOH	kNN acc.
Adaptive [29]	193%	16%	0.67
Decoy [23]	100%	39%	0.25
BuFLO [8]	145%	180%	0.08
Supersequence [31]	222%	112%	0.05
Tamaraw [5]	103%	140%	0.05
WT (this work)	**31%**	**34%**	**0.28**

We can see from Table 4 that WT has a markedly smaller bandwidth overhead (BWOH) and time overhead (TOH) than many of the previous attacks, and it is still able to defeat kNN. Across our data set, the bandwidth overhead of WT is $31\% \pm 16\%$ and its time overhead is $34\% \pm 5\%$; different cell sequences vary significantly in bandwidth overhead but not time overhead. BuFLO, Supersequence, and Tamaraw are able to further decrease kNN accuracy (0.05 to 0.08) compared to WT (0.28), but this effectiveness comes at a high cost in overhead. kNN's higher accuracy against WT is not practically meaningful: nevertheless, the attacker cannot identify accesses to sensitive pages under WT due to the base rate fallacy. For WT, any cell sequence always looks as if it could have come from at least two different web pages due to burst molding, which means that no WF attack can reach an accuracy above 0.5. We develop this notion further in Section 5.4.

Tamaraw, Supersequence, and WT are all tunable: each defense can decrease its own time overhead by increasing its bandwidth overhead and vice versa. Furthermore, each defense can increase either overhead to increase the effectiveness of the defense against attacks. A proper comparison of these defenses requires further analysis. We focus on Tamaraw as it has a lower overhead than Supersequence.

We investigate the trade-off between time overhead and bandwidth overhead. To do so, we fix the effectiveness of Tamaraw and WT to be the same against attacks in general (see Section 5.4 for details on how we compute this). For Tamaraw, the trade-off is achieved by varying the fixed intercell times. For WT, the trade-off is achieved by changing which cell sequences to choose in burst molding. We can prefer cell sequences that minimize bandwidth overhead at the cost of time overhead and vice versa. We plot the results in Figure 2. We find that the range of possible overheads for WT is quite small compared to Tamaraw. Half-duplex communication induces a 30% time overhead in our experiments, so that is the minimum value for WT. While the overhead of Tamaraw can vary significantly, its range of both bandwidth

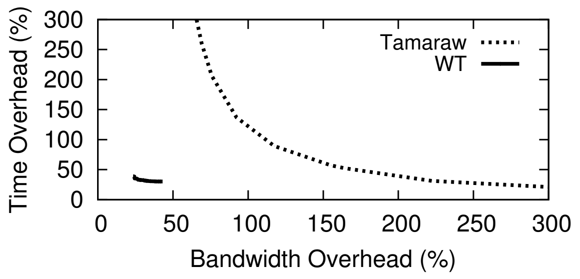

Figure 2: Bandwidth and time overhead for Tamaraw and WT.

and time overhead is in any case much higher than that of WT. To reach a bandwidth overhead less than 100%, for example, a time overhead over 150% is required, which is a large increase in page load time.

To investigate the trade-off between overhead and effectiveness, we need a general notion of effectiveness for all attacks, not just any given attack. We next develop such a notion and show that WT is effective against all WF attacks in general.

5.4 Defending against any classification attack

Observing that many older defenses have not proven effective against newer attacks, authors in the field [4, 31] have suggested that a defense should be designed to be effective against all possible WF attacks. To do so, the output cell sequences of some web pages should be *exactly the same* as some other web pages. To be specific, the cell sequences should be the same length, and the timing, direction, and size of all cells should be the same.[3] If this is achieved, then no attacker can distinguish between those web pages, independent of the classification mechanism they use.

The above is achieved in both Tamaraw and WT. We compare Tamaraw and WT in terms of their effectiveness against all possible WF attacks.

5.4.1 Maximum Attacker Accuracy

Borrowing terminology from the k-anonymity literature, we say that two cell sequences s, s' belong to the same collision set $C(s)$ if they become the same sequence after applying the defense. They may come from different web pages; we denote the page a cell sequence comes from as $Page(s)$. An effective defense's objective is to cause cell sequences to collide. We measure the effectiveness by defining a notion of Maximum Attacker Accuracy (MAA). The MAA of a cell sequence is equal to:

$$MAA(s) = \frac{|\{s' \in C(s) | Page(s') = Page(s)\}|}{|C(s)|}$$

The MAA describes an attacker who, seeing that they cannot distinguish between any of the cell sequences in the collision set, decides to simply randomly guess which page it is. On the other hand, if all cell sequences in the collision set belong to the same page anyway, the attacker's guess will be exactly correct. The attacker maximizes classification accuracy in the sense that they know exactly which page each cell sequence belongs to ($Page(s)$ is known to the attacker for all s). No classifier's accuracy can exceed the MAA; the lower the MAA, the more effective the defense. We thus favor the MAA as an intuitive, attack-agnostic metric for measuring the minimum effectiveness of a defense. Later, in Section 6.2, we expand on the MAA by investigating WT in an open-world scenario with different page visit rates; for now, we evaluate WT on a simpler MAA.

It is easy to see that the MAA of Walkie-Talkie is 0.5. Each cell sequence is in a collision set with exactly one other cell sequence from a different page due to burst molding. Furthermore, since the decoy page selection mechanism is symmetric (Section 4.3.2), the collision set does not reveal which cell sequence is the true cell sequence. However, if we increase the number of colliding cell sequences, the MAA can lower further. We develop this idea next.

5.4.2 Maximum Attacker Accuracy of WT

In the context of WT, the MAA is that of an attacker who knows exactly which two pages can be the decoy page and the real page, but not which is which. In other words, he resorts to guessing one out of two pages. We can decrease his MAA by molding towards the supersequence of several decoy cell sequences, not just one decoy cell sequence.

The greater the number of cell sequences chosen, the greater the overhead. We investigate the MAA of WT and compare it with Tamaraw. We show the results in Figure 3, plotting MAA against bandwidth overhead. WT is generally more efficient even if the user desires a very low MAA. The time overhead of WT goes up to 45% for the values in this graph, while it increases much more quickly for Tamaraw, from 130% to 350%.

WT has another advantage over other WF defenses: any defended cell sequence could have come from many different web pages. This is because any subsequence of a defended cell sequence could have been the original undefended cell sequence. Not all possibilities are equally

[3]We do not need to ensure that the cells were received at the same time including network noise; we only need to ensure that the cells were attempted to be sent at the same time, as any timing difference then would only indicate network noise and reveals no information about the cells themselves.

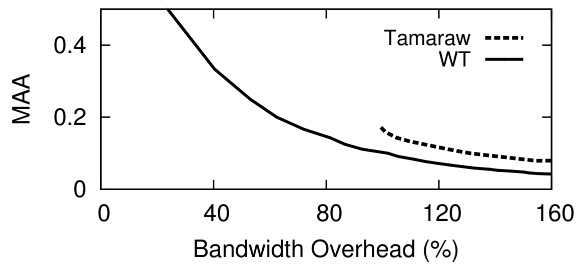

Figure 3: Bandwidth overhead and MAA for Tamaraw and WT across a range of parameters. No WF attack can achieve a classification accuracy above the MAA.

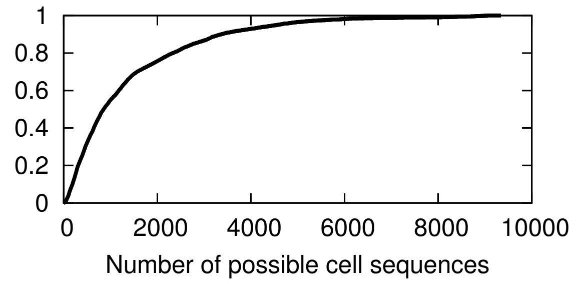

Figure 4: Cumulative distribution frequency graph of WT collision set sizes. A collision set of a defended cell sequence is the set of undefended cell sequences that could have generated it when the defense is applied.

likely: burst molding attempts to minimize overhead, so from the attacker's perspective, the true cell sequence is not likely to be much smaller than the observed cell sequence. Nevertheless, this observation produces a confusing effect on the attacker that has not been accounted for in the MAA; that is, a realistic attacker's accuracy is likely to be lower than the MAA.

We evaluate this effect on our closed-world page set of 100 pages and 100 instances each. For each defended cell sequence, we calculate the number of possible undefended cell sequences from other web pages that could have generated it. We call this the **collision set size**. The maximum collision set size is therefore 9900. We show the cumulative distribution frequency graph in Figure 4. There was only a .1% chance that the collision set size was smaller than 10 (it was always at least 2 because of burst molding), and a 4% chance that it was smaller than 100. The median collision set size was 860. We contrast this with Tamaraw, where on our data set there was a 2% chance that the collision set size was smaller than 10 and a 13% chance it was smaller than 100; the largest collision set size was 795. The attacker's ability to rule out possible web pages given a defended cell sequence is much more limited under WT.

6 Extensions of Walkie-Talkie

In this section, we present several extensions of Walkie-Talkie to defeat three WF attackers that are more advanced than that of previous work. In Section 6.1 we describe multi-page attackers, who understand the relationship between several pages of the same site and can determine when the client is on the same site. In Section 6.2 we describe attackers who know that the client visits pages at different base rates, and can estimate this base rate. In Section 6.3 we investigate attackers that can use timing information to defeat Walkie-Talkie. We show that, with some modifications, WT can effectively defend clients at little extra cost against all of these advanced attackers.

6.1 Defending against multi-page classification

In Section 5, we analyzed Walkie-Talkie against an attacker who classifies pages one at a time, independently of any other page. A realistic attacker could leverage his knowledge of the link structure of web sites to achieve greater accuracy. For example, if the attacker knows a priori that two web page accesses came from the same site, then the attacker can more accurately identify what site that is.

Defending against multi-page classification critically relies on the ability to specify which non-sensitive decoy page to use for each sensitive page. With this feature, we can specify non-sensitive pages from the same site as decoys when the client is visiting sensitive pages from the same site. BuFLO-based defenses are unable to specify decoy pages, while Supersequence and Glove must suffer significant overhead to do so. However, WT is able to choose decoy pages with great efficiency. WT is thus well suited as a defense against multi-page classification. We modify WT so that it chooses decoy pages more cleverly. When the client is visiting sensitive pages from the same site, WT also mimics non-sensitive pages from the same site, each one of which is likely to lead to the next.

With the above modification, WT will succeed in defending clients against multi-page attacks, which no previous WF defense has done. To demonstrate this, we will evaluate its overhead and Maximum Attacker Accuracy against multi-page attackers. We expect the overhead to be higher than before, because the client has less freedom of choice in page selection.

We experiment by configuring our Tor Browser client to randomly follow links on each of Alexa's top 100 sites. Unfortunately, we do not know the true probabilities with which real clients visit links from Alexa's top 100 sites, so we choose the next link uniformly randomly from the set of all links on the page. The client stops after 10 page loads. Then, we test the bandwidth and time overhead of a client attempting to decoy random sensitive pages with those page loads. We find that, maintaining an MAA of 0.5, the bandwidth overhead necessary to defend against a multi-page attacker increases from 31% to 53%, and the time overhead increases from 34% to 42%. The increase is small, and demonstrates that a client can effectively defend herself against multi-page attacks as well, with no decrease in minimal defense effectiveness.

6.2 Incorporating prior knowledge

For analytical simplicity, our experiments assumed a client that visits all pages with the same likelihood; to our knowledge, all other works in website fingerprinting make this assumption. Realistically, a client would visit pages with different probabilities, and the attacker may have prior knowledge of such a distribution. Here, we remove the previous assumption and adopt a model for estimating page likelihood, assuming that the attacker knows the client's distribution fully. We examine how this affects Walkie-Talkie.

We obtain basic estimated page view data for Alexa's top 10,000 sites from StatShow, and perform least-squared approximation on the logarithm of the number of page views. We attempted to approximate the number of page views with the following function

$$Views = a \cdot e^{bx} \cdot (x+1)^{c}$$

In the above, a, b, and c are parameters, and x being the index of the page (1 being most popular). We obtain the parameters by performing Levenberg-Marquardt least-squared approximation on the logarithm of the above function, resulting in $a = 36000$, $b = -0.000083$, $c = -1.0$. However, we found that the number of page views dropped precipitously near the end of our data set, rendering parameter estimation inaccurate. We believe this is because our list of top sites was incomplete at the end of the list. Instead of trying to fit all of our data, we fit the top 5,000 sites and then extrapolate. The resultant curve had a mean squared error of 0.0002 on the logarithm of the number of page views.

We simulate clients that visit pages with probability based on this curve, with no limit on the index of the page. Our model suggests that 57% of all page views are in the top 100 sites, and 40% of all page views are in the next 10,000 sites. We use the former set as non-sensitive decoy pages and the latter as monitored sensitive pages. Considering an ambitious, powerful attacker who is always capable of identifying the potential decoy and sensitive page in a WT-protected page access (but not which is which), the attacker can achieve a precision of 41% by simply guessing that all pages are sensitive (with a recall of 100%). In a more realistic scenario when the attacker

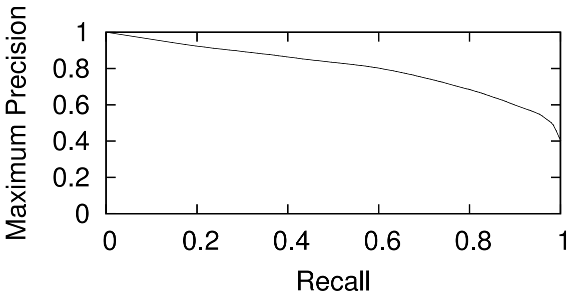

Figure 5: Maximum precision/recall graph for an attacker on Tor defended by WT, after incorporating page likelihood.

is interested in much fewer than 10,000 pages, the maximum precision would be proportionally lower.

We draw the attacker's precision/recall curve by having him cleverly choose to identify sensitive pages in decreasing order of precision, and gradually increasing the set of such pages he was willing to classify. This gives the attacker the maximum precision at each level of recall. We draw the graph in Figure 5. For instance, we find that at 25% recall, the attacker has a maximum precision of 90%. Even with such a low recall, the attacker frequently makes mistakes in identifying sensitive pages. We can contrast this with kNN, which can achieve a precision of 99% with a recall of 80% on a non-defended Tor data set [31]. The attacker's precision does not change even if the attacker had prior information indicating that the client is not visiting certain monitored pages, as long as the visit rate of other pages is unchanged.

We consider a page-view-sensitive variation of WT where we also choose decoy pages based on the popularity of the page, not just the potential overhead. This method would come with a penalty to the overhead. We take the value of maximum precision for at least 25% recall, and we we plot the graph of maximum precision to bandwidth overhead in Figure 6. (Time overhead increased slowly from 34% to 42% within the range of this graph.) The maximum precision starts at 90% and then drops sharply to 64%. (The minimum precision of simply randomly guessing if each page is sensitive is 41%.) However, as we increase the weight for page popularity further, we see that the maximum precision increases counter-productively. This represents the case where the client starts choosing only the first few most popular pages, which limits her set of potential decoys, weakening her defense.

6.3 Intercell timing

WT may have a subtle timing leak: the incoming cell rate may leak information about the destination web page—more precisely, the number of servers that are sending in-

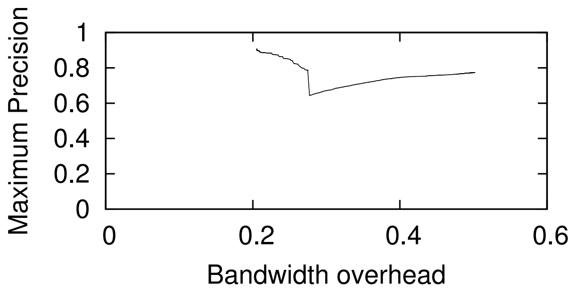

Figure 6: Maximum precision/bandwidth overhead graph for an attacker on Tor defended by a page-view-sensitive variation of WT. The variation decreases precision further for a small increase in bandwidth overhead.

formation simultaneously, and their possible processing times before starting to send the page data. For outgoing cells, timing leaks no information for WT, because there is only one client and half-duplex communication ensures that the client is dumping all the requests she can send as quickly as possible, after which she falls silent. In this section, we first argue with empirical evidence why the incoming intercell timing leak of WT may not be practically usable by any attacker. Nevertheless, we then show how WT can be modified to cover any possible incoming intercell timing leak. Despite the lack of empirical evidence that this timing leakage can be leveraged by any attacker, we provide such a modification to preserve the theoretical guarantees of WT against future WF attacks that may more cleverly use intercell timing.

Is timing useful for classification?

The results of this work have already suggested that intercell timing is not useful: in Section 5.2 and Section 5.3, we allowed WT to leak intercell timing, and WT was nevertheless able to efficiently defeat known attacks. In fact, WF researchers tend to avoid the use of intercell timing in general: out of fourteen known WF attacks we surveyed, we found that only three attacks used intercell timing: the two oldest WF attacks [2, 29] (both are significantly less effective than newer attacks on Tor), and kNN [31]. We specifically saw in Table 3 that intercell timing does not aid classification in kNN either. We ran a further classification test using kNN only on extracted intercell timing values of the top 100 pages, and achieved a 0.5% TPR.

We suggest that this is because intercell timing is highly inconsistent for the same site, but the distribution of timings is similar across different sites. Network conditions fluctuate rapidly as proxies need to be rotated frequently to safeguard anonymity. We constructed kernel density estimators using Scott's rule [28] on intercell timing, and found that the resulting probability density functions overlapped significantly. Experimentally, we

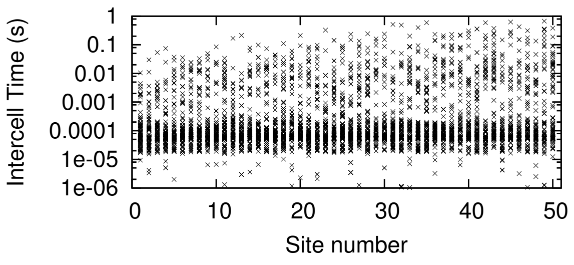

Figure 7: 100 random intercell times from each of 50 top pages. Each cross represents an intercell time. Note that the y-axis is logarithmic.

found that the attacker could only achieve a maximum 2% accuracy on the top 100 pages by choosing the most likely page for each sampled intercell timing value.

To illustrate this point visually, we plotted 100 random intercell times from each of the top 50 pages[4] in Figure 7, in ascending order of mean intercell times. Figure 7 suggests that intercell times vary significantly, but their patterns are not noticeably different across different sites. This shows that individual intercell times are not correlated with the true page of a cell sequence.

Equalizing intercell timing

We have nevertheless designed an extension of WT to hide all timing information, though this comes at the cost of a greater bandwidth and time overhead.

One solution would be to have the proxy behave the same way as the client: it queues all received cells in each burst until the servers have sent all of their data, and sends them all at once back to the client. In this case, timing would contain no information, and this can be implemented with a small time overhead and no bandwidth overhead. However, this implementation may not be practical, because it would require proxies to read client cells to determine when bursts end.

Our timing fix is inspired by a similar mechanism in Tamaraw. We choose a fixed cell rate $r_{control}$ such that whenever it is the proxy's turn to send data, the proxy attempts to deliver $r_{control}$ incoming cells per second. If there is no data to send when a cell is due, the proxy generates a dummy cell, which will be dropped by the client. This covers the incoming intercell timing leak as the intercell time will always be $r_{control}$ for incoming cells. Varying $r_{control}$, we evaluate the added overhead of timing control in Figure 8. For example, we can equalize intercell timing at a cost of 50% bandwidth overhead and 36% time overhead.

We can use the same dummy cells described in Section 4.3 for both burst molding and equalizing intercell timing, without compromising either objective. This

[4]We used 50 pages instead of our full 100 pages so that the graph would be clear.

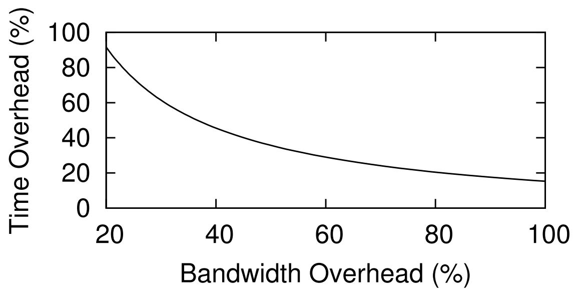

Figure 8: Possible bandwidth and time overhead cost for equalizing intercell timing, obtained by varying $r_{control}$.

means that, effectively, the bandwidth overhead values for timing control and burst molding do not add together in WT; instead, the maximum of the two becomes the bandwidth overhead of WT. The overhead of WT with intercell timing equalized would be 50% bandwidth overhead and 66% time overhead, which is still much lower than known defenses (Table 4).

7 Conclusion

In this paper, we presented Walkie-Talkie: a flexible, easy-to-use defense with low overhead that can defend web clients against all website fingerprinting attacks. Walkie-Talkie consists of two components: half-duplex communication and burst molding. Half-duplex communication produces burst sequences that are concise and easy to manipulate, which allows burst molding to mimic non-sensitive web pages at minimal overhead. Walkie-Talkie is highly effective against all known attacks at overhead costs much lower than all known effective defenses. Furthermore, it is capable of defending against all possible WF attacks, because pairs of sensitive and non-sensitive web pages will be molded to the same cell sequence under WT. We have implemented Walkie-Talkie so that it functions on the Tor client and Tor nodes: in general, it can be implemented on any proxy network (such as VPNs).

We also considered advanced attackers beyond previous work in website fingerprinting, who are able to leverage site link information, page visit rates, and timing information to strengthen their attacks. Walkie-Talkie can defend against all these types of attacks effectively, as it gives the client the freedom to choose which pages to use as decoys. It remains to be seen whether Walkie-Talkie would be useful as well against other advanced attacks, such as active adversaries and dynamic content identification.

Acknowledgements

We thank our shepherd, Scott Coull, for his help in shaping the final version of this paper, and the anonymous reviewers for their helpful comments. This work benefitted from the use of the CrySP RIPPLE Facility at the University of Waterloo. We thank NSERC for grant RGPIN-341529.

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] D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proceedings of the 12th USENIX Security Symposium*, 2003.

[4] X. Cai, R. Nithyanand, and R. Johnson. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *Proceedings of the 13th ACM Workshop on Privacy in the Electronic Society*, 2014.

[5] X. Cai, R. Nithyanand, T. Wang, I. Goldberg, and R. Johnson. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, 2014.

[6] X. Cai, X. Zhang, B. Joshi, and R. Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pages 605–616, 2012.

[7] 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`, 1998.

[8] 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.

[9] P. Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, pages 1–18, 2010.

[10] R. Fielding and J. Reschke. Hypertext transfer protocol (http/1.1): Message syntax and routing. 2014.

[11] G. Greenwald. NSA Prism program taps in to user data of Apple, Google and others. `http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data`, June 2013. Accessed Apr. 2015.

[12] J. Hayes and G. Danezis. k-Fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *Proceedings of the 25th USENIX Security Symposium*, 2016.

[13] 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.

[14] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, 2014.

[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] 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.

[17] 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 *Proceedings of the 18th Network and Distributed Security Symposium*, 2011.

[18] 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*, pages 143–163, 2014.

[19] S. J. Murdoch and G. Danezis. Low-Cost traffic analysis of Tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183–195, 2005.

[20] R. Nithyanand, X. Cai, and R. Johnson. Glove: A Bespoke Website Fingerprinting Defense. In *Proceedings of the 13th ACM Workshop on Privacy in the Electronic Society*, 2014.

[21] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. The spy in the sandbox: Practical cache attacks in javascript and their implications. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, 2015.

[22] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel. Website fingerprinting at internet scale. In *Proceedings of the 23rd Network and Distributed System Security Symposium*, 2016.

[23] 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.

[24] M. Perry. Experimental Defense for Website Traffic Fingerprinting. `https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting`, September 2011. Accessed Feb. 2015.

[25] M. Perry. TBB's Firefox should use optimistic data socks handshake variant. `https://trac.torproject.org/projects/tor/ticket/3875`, August 2011. Accessed Apr. 2015.

[26] M. Perry. A Critique of Website Traffic Fingerprinting Attacks. `https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks`, November 2013. Accessed Feb. 2015.

[27] R. Schuster, V. Shmatikov, and E. Tromer. Beauty and the burst: Remote identification of encrypted video streams. 2017.

[28] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, 2015.

[29] V. Shmatikov and M.-H. Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *Computer Security–ESORICS 2006*, pages 18–33. 2006.

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

[31] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[32] T. Wang and I. Goldberg. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, pages 201–212, 2013.

[33] T. Wang and I. Goldberg. On Realistically Attacking Tor with Website Fingerprinting. In *Privacy Enhancing Technologies*. Springer, 2016.

[34] 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.

A Removing the need for client information

One limitation of burst molding is that the client needs to know the burst sequences of some non-sensitive pages. While it is highly practical to deliver such information to clients on an anonymity network like Tor (see Section 4.3.1), we have also designed a variation of Walkie-Talkie that is useful on networks where there may not be a party that can deliver burst sequence information.

In this variation of Walkie-Talkie, burst molding is *random*: instead of adding cells according to the supersequence of sensitive and non-sensitive pages, we add cells randomly. We refer to this variation of Walkie-Talkie as Random-WT. Random-WT is less efficient, but it can also defend against all possible WF attacks.

A.1 Design of Random-WT

Given a cell sequence $s = (b_1, b_2, b_3, ..., b_{|s|})$ with $b_i = (b_{i+}, b_{i-})$, we apply defense D as follows to produce $D(s)$:

1. Padding real bursts: From two uniform distributions X_{i+} and X_{i-}, we draw x_{i+} and x_{i-} respectively, and add them to b_i, such that $\hat{b_i} = (b_{i+} + x_{i+}, b_{i-} + x_{i-})$.

2. Adding fake bursts: From two uniform distributions X_{id+} and X_{id-}, we draw x_{id+} and x_{id-}, and generate a new fake burst $\hat{b_i} = (x_{id+}, x_{id-})$. In a fake burst, all outgoing and incoming cells are fake cells. We add fake bursts at random with probability p_{fake} before each real burst of cells.

Random-WT is therefore defined by the bounds of the uniform distributions X_{i+}, X_{i-}, X_{id+}, X_{id-}, as well as the probability p_{fake}. We chose uniform distributions after preliminary experiments and analysis indicated that uniform distributions are highly efficient at defending burst sequences. The freedom of choice allows Random-WT to be tunable (i.e., a client may wish to increase collision rate by increasing overhead).

The fact that any burst in an observed cell sequence is equally likely to be fake is a powerful feature of Random-WT. In practice, we found that many cell sequences have multiple bursts with few cells and one or two large bursts with many cells. Random-WT covers the position of large bursts in the cell sequence, so that they cannot be leveraged by the attacker.

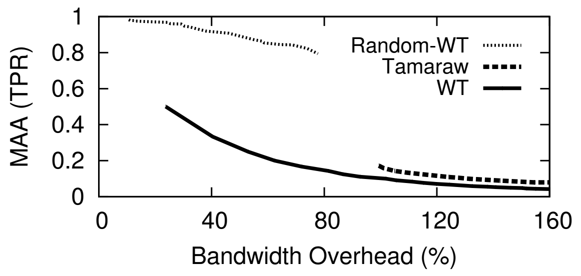

Figure 9: Bandwidth overhead and MAA for Random-WT, WT and Tamaraw across a range of parameters. No WF attack can achieve a classification accuracy above the MAA.

Fake bursts should be similar in length to real bursts so as to maximize collision; we do not want the attacker to be able to distinguish between real bursts and fake bursts with high accuracy. In our experiments, we set the lower bound of the uniform distributions for X_{i+}, X_{i-}, X_{id+}, and X_{id-} to be 0. This minimizes overhead without affecting effectiveness. We set X_{id+} and X_{id-} to fit the observed burst sizes of real cell sequences. Then, we vary the maximum range of X_{i+} and X_{i-}, as well as p_{fake}, to obtain a range of overhead and effectiveness values, which we present below.

A.2 Experimental analysis of Random-WT

We analyze the MAA of Random-WT using the same experimental methodology in Section 5.4. We draw Figure 9 by taking Figure 3 and adding a line for Random-WT to compare with basic WT and Tamaraw. Figure 9 does not plot the time overhead, which is around 30% throughout the graph for both Random-WT and WT, and ranging from 130% to 350% for Tamaraw.

Though Figure 9 shows that the MAA of Random-WT is worse than both WT and Tamaraw, it is still significant enough to confuse an attacker, especially if the attacker needs a low false positive rate (for example, when attempting to identify accesses to rare pages). An advantage of both WT and Random-WT over Tamaraw is that they are not sensitive to network conditions; previous work has shown that Tamaraw performs worse than expected if network conditions are not correctly predicted [4].

B Publication of code and data

To ensure that our results can be reproduced, we publish the following:

- Our implementation of Walkie-Talkie: the Firefox code that modifies the browser to enable half-duplex communication, the Tor code that modifies the Tor client to enable molding, and our experiment code for WT.
- Our experimental data sets: the cell sequences we collected over Tor with and without half-duplex communication.
- Our implementations of previous attacks and defenses.

The code and data are available at `https://crysp.uwaterloo.ca/software/webfingerprint/`.