Walkie-Talkie: An Effective and Efficient Defense against Website Fingerprinting

Tao Wang
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
t55wang@cs.uwaterloo.ca

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

ABSTRACT

Website fingerprinting 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. Effective defenses against website fingerprinting hamper user experience due to their large bandwidth overhead and time overhead, requiring more than a half minute to load a page on average. In this work we propose a new defense against website fingerprinting, `Walkie-Talkie`, with a small overhead that can confuse even a perfectly classifying attacker. `Walkie-Talkie` modifies the browser to communicate in half-duplex mode rather than the usual full-duplex mode, thus restricting the feature set available to the attacker. We then add random padding to further confuse the attacker. With `Walkie-Talkie`, at a bandwidth overhead of 32% and time overhead of 9%, the perfect attacker's false positive rate exceeds 5%; at a bandwidth overhead of 55%, the perfect attacker's false positive rate exceeds 10%. Our defense therefore allows web-browsing clients to defend their privacy against website fingerprinting both effectively and efficiently.

1. INTRODUCTION

Website fingerprinting allows a local, passively observing eavesdropper to determine which web page a client is visiting by observing the sequence of packets. The attacker uses various packet sequence features, such as packet counts, packet order, packet directions, and unique packet lengths to classify the web page. [4] Website fingerprinting attacks require no extra capabilities, small computational cost, and carry little risk of detection. Previous authors have found that even privacy technologies that hide packet contents, destinations, and sizes, such as VPNs, IPsec, and Tor, are susceptible to website fingerprinting. [5, 9, 11, 12, 15, 21] 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 website fingerprinting in some way.

Website fingerprinting is a well-established threat to privacy in the literature [7, 10, 15], as well as in practice: Tor, a popular anonymity network, has implemented two variations of a website fingerprinting defense [16, 18]. However, these defenses are not effective [5, 21]. Researchers have proposed alternative defenses, but these defenses are either ineffective against newer attacks [4] or carry a very large overhead [3, 7, 14, 21]; we describe previous website fingerprinting work in detail in Section 2.

We refer to defenses that are effective against all classification attacks as *general defenses*. With known general defenses, the overhead is very large: clients can expect a mean web page load time between half a minute and a minute, which is frustrating. Therefore, we want to design a defense against website fingerprinting with the following design goals:

1. General: The defense succeeds against all possible classification attacks by causing *collisions*, where the defense outputs the same packet sequence given input packet sequences from different web pages. Even a perfect attacker would not be able to tell which page it came from.

2. Easy to use: The client does not need to configure the defense. The defense is ready to use out of the box, and can be deployed incrementally as it does not depend on other clients using the same defense.

3. Decentralized: The defense should not require some central server, with a shared database, to operate. We want to match the decentralized model of anonymity networks.

In this paper we present `Walkie-Talkie`, which meets these criteria. Our defense is effective against all attacks with a significantly lower (and tunable) overhead compared to previous work. `Walkie-Talkie` modifies the client's behavior when browsing by enforcing half-duplex communication, 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. Under our defense, the client only sends requests after the web servers have satisfied all previous requests. The client and servers both send data in interleaving bursts of incoming and outgoing packets. We describe the implementation of half-duplex communication in the browser in Section 3.

Half-duplex communication significantly reduces the feature set available to the website fingerprinting attacker: only the number of packets in each burst is available. Further, as we will see in our analysis and experiments, half-duplex communication adds only a small amount of time overhead, as it does not change the number of round trips needed to load a page, and the round-trip times are a significant component of the page-load time in anonymity networks such as Tor. We then add dummy packets to pad the number of packets in each burst to further confuse the attacker, as described in Section 4. Using the evaluation methodology described in Section 5, we evaluate `Walkie-Talkie` in Section 6 on a data set collected over Tor. We show that known attacks, as well as a perfect attacker, are unable to perform website fingerprinting against packet sequences under `Walkie-Talkie` with high accuracy, and that our defense has a significantly lower overhead compared to known defenses. In Section 7 we describe and evaluate three variants of `Walkie-Talkie` that do not satisfy some of the design goals above. In Section 8, we discuss how our results can be reproduced and timing analysis attacks on our system. We conclude in Section 9.

2. RELATED WORK

Researchers have identified website fingerprinting (WF) as a potential privacy leak since 1998 [6]. 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 Internet users en masse [8]. As a result, Tor currently employs a WF defense [16]. 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. [5, 9, 11, 12, 15, 20, 21] 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 web pages the attacker knows, and the attacker must distinguish packet sequences coming from those pages. In the open-world scenario, testing packet sequences could originate from non-monitored web pages outside of the list and unknown to the attacker.

Over time, researchers have demonstrated increasingly accurate and noise-tolerant attacks using different classification tools. 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 other works [4, 21] for a more detailed discussion of the specific workings of each WF attack and how they have evolved.

2.2 Defenses

In this section we discuss two types of defenses: *limited* and *general* defenses. Limited defenses succeed against specific WF attacks; general defenses succeed even against an attacker who performs classification perfectly accurately. The new defense we propose, `Walkie-Talkie`, is a general defense.

2.2.1 Limited defenses

The earliest WF defenses were designed against specific, effective attacks. Wright et al. (2009) published traffic morphing [23], 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 Liberatore and Levine's attack [11] (2006), 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 attack, which uses packet ordering as a feature. [21]

Luo et al. (2011) published HTTPOS (HTTP Obfuscation) [13]. They implemented the defense on the client side using specific 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, 11, 20], but other researchers have also found that it is not a successful defense against several newer attacks [5, 21].

Tor has implemented another WF defense [16] in response to a WF attack by Panchenko et al. [15]. Tor's defense uses HTTP pipelining[1] 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 [18] recently in response to newer attacks, but both versions of the defense have little effect on the accuracy of known attacks [5, 21, 22].

[1]HTTP pipelining is implemented in modern browsers but it is not enabled by default in IE, Firefox, and Chrome. Tor Browser, which is based on Firefox, enables pipelining because of this defense, but the defense does not take effect if the web server does not support pipelining, which is still common [13].

2.2.2 General defenses

The above shows us that defenses designed only to defeat older attacks often fail against newer attacks. The client cannot fully trust limited defenses to carry privacy-sensitive traffic, especially against invested adversaries that are willing to update their attacks.

Recently, researchers have developed *general* defenses: defenses that would succeed against any possible classifier. In other words, different web pages should, with sufficient likelihood, produce the exact same packet sequence.

Dyer et al. (2012) described BuFLO [7], in which the client sends and receives traffic at a fixed, constant rate. The traffic is padded with dummy packets that contain no information if there are no real packets to send. The duration of each page load is padded up to 10 seconds if it is lower; there is no further padding if it is higher. Dyer et al. found that this defense still failed often against WF attacks. Cai et al. (2014) argued that this is because the duration frequently exceeded 10 seconds, so it is not padded, thus revealing the approximate size of the page [4]. They showed that Tamaraw, their modification of BuFLO, can fix this problem while significantly reducing overhead. Similarly Wang et al. (2014) presented Supersequence [21], finding that supersequences offer the optimal-bandwidth solution if the defense is only applied on the wire, but searching for such a solution is still difficult.

2.3 Moving Forward

The known general defenses—Tamaraw and Supersequence—use constant-rate packet delivery, which has several problems:

1. Overhead. A major barrier towards implementation of general defenses is that the bandwidth overhead and time overhead (these terms are defined in Section 5.3) are both very large. Under constant-rate packet delivery, both the client and the servers send many dummy packets. Researchers have reported a 90% to 180% bandwidth overhead [3, 4, 21] and a 170% to 240% time overhead [3]. For Tor, the time overhead implies that loading a single web page will take more than half a minute. We show that these defenses are overkill in Section 6.3.

2. Inflexible packet rate. Cai et al. [3] have found that the fixed packet rate demanded by Tamaraw and Supersequence causes poor behavior in congested networks, especially when other protocols are also demanding bandwidth, since there is no congestion control in constant-rate packet delivery. The authors have suggested that varying the packet rate depending on congestion would improve its network performance, but it is not clear if the resulting defense would still be general.

In this work we show that by using half-duplex communication, `Walkie-Talkie` addresses both of these problems: it is a general defense with low overhead and no restriction on packet rate.

3. IMPLEMENTING HALF-DUPLEX COMMUNICATION

Under half-duplex communication, the web-browsing client and the servers send alternating bursts of packets: the client sends the first outgoing burst, the server sends the first incoming burst, the

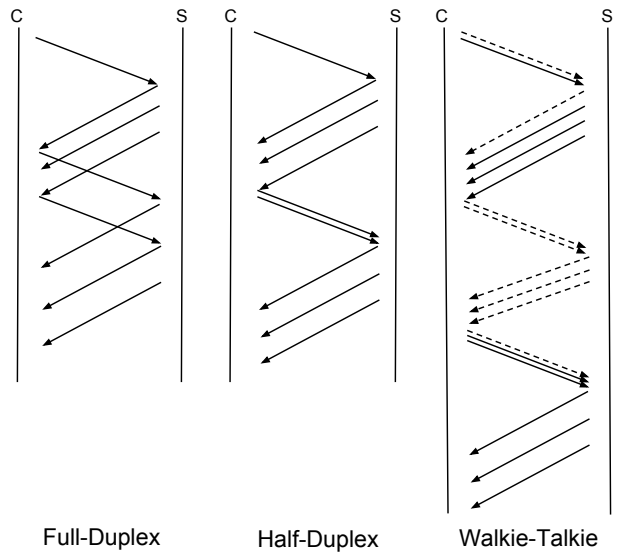

Figure 1: Client-server connection schema for Walkie-Talkie. Under normal HTTP, clients (C) send HTTP requests to the server (S) as soon as possible. With half-duplex communication, clients send HTTP requests only after all previous requests have been satisfied by the server in accordance with half-duplex communication. Walkie-Talkie uses half-duplex communication with extra padding packets (dashed lines) as well as dummy burst pairs.

client replies with the second outgoing burst, and so on. We illustrate how half-duplex communication affects HTTP in Figure 1.

Half-duplex communication reduces the attacker's potential feature set to only a series of burst traffic sizes, which makes WF difficult. Previous general defenses relied on constant-rate packet delivery. The advantages of half-duplex communication as a WF defense compared to constant-rate packet delivery are as follows:

1. Small time overhead. Constant-rate packet delivery has a high time overhead because it required the packet rate to be low in order to minimize the need for dummy packets and thus reduce bandwidth overhead. In contrast, half-duplex communication does not impose any limit on the packet rate.

2. No bandwidth overhead. Half-duplex communication has no bandwidth overhead; the only bandwidth overhead in our defense, `Walkie-Talkie`, comes from the random padding, described later in Section 4.

3. Implementation. Cai et al. found barriers to implementing constant-rate packet delivery [3] as mentioned in Section 2.3. However, half-duplex communication is easy to implement; we have already done so, and we next provide details on our implementation.

3.1 How browsers work

In this section we will briefly describe how browsers use persistent connections to load data from a web server. We will use the terminology present in Firefox, although the underlying operations are the same for all modern browsers.

During web browsing, clients request data from the server (or post data to the server) with transactions, which start with an HTTP request from the client. To dispatch transactions, the browser creates or re-uses long-lived TCP/IP connections (up to a preset maximum number of connections). When transactions are complete, the browser may close the attached connection, or keep them alive as idle connections in order to dispatch further transactions to the same server.

If the browser cannot dispatch a transaction (generally due to a limit on the number of connections per server or in total), the browser stores the transaction in a pending transaction queue. When any connection dies or becomes idle, the browser enumerates the pending transaction queue in an attempt to dispatch every transaction (sometimes by creating new connections). During the enumeration process, the browser may re-use idle connections or close them to make room for new connections to other servers.

Within Firefox, a connection manager keeps track of connections. The connection manager owns the pending transaction queue, counts connections, and enforces limits on the number of connections, amongst other responsibilities. To enforce the limit on the number of connections, the connection manager decides when to queue a transaction and when to enumerate the transaction queue to dispatch transactions. We implement half-duplex communication by modifying the connection manager.

3.2 Implementing half-duplex communication

We add two states to the connection manager 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. We explain each below.

The connection manager starts in the *walkie* state. When the client starts any transaction in the *walkie* state, the connection manager dispatches the transaction immediately, and the connection manager switches to the *talkie* state. After each page has finished loading, when there are no pending transactions left, the connection manager will return to the *walkie* state.

In the *talkie* state, the connection manager is currently loading a page. The connection manager always queues new transactions in this state; it never dispatches transactions immediately. Furthermore, the connection manager does not enumerate the pending transaction queue for dispatching whenever any connection dies or become idle. Rather, the connection manager only enumerates the transaction queue for dispatching when there are no active connections left (i.e. all connections have died or become idle). At this point, the connection manager first attempts to dispatch the entire transaction queue. If the queue is empty, the connection manager returns to the *walkie* state; page loading has stopped.

We justify why the above states implement half-duplex communication in two steps. First, the client should only be talking when the web servers are not talking. Second, the web servers should only be talking when the client is not talking.

1. If the client is talking (dispatching transactions): In the *walkie* state, there are never any active connections, and the client is allowed to dispatch new transactions at any time. In the *talkie* state, the client only attempts to dispatch new transactions when there are no active connections left. In both cases, when the client dispatches a transaction, there are no active connections; since an HTTP server does not initiate contact with the client again after the connections have died or become idle, the server is not talking.

2. If the servers are talking (responding to transactions): The client sends all transaction requests at the same time, when the pending transaction queue is enumerated for dispatching. The server then responds with content for each transaction. During this time, the client waits for responses for all these transaction requests and does not attempt to dispatch new

transaction requests until all requests have been satisfied, so the client is not talking.

We note that in the above proof (specifically in the second portion), we made the assumption that transaction dispatching is nearly instantaneous as compared to the round-trip time, but this is not normally true. This is because dispatching a new transaction may involve two steps: first, establishing a TCP connection, and second, sending the HTTP request. 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 perform these two steps (establish a connection and sending the HTTP request) in two bursts rather than one burst. However, there is a more efficient way to do so, as described below.

3.3 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. For multi-hop anonymity networks, 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. [17] 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 on the socket level; it does not involve the connection manager. Normally, after sending a connection establishment request, the socket waits for an acknowledgment by the server before informing the connection manager that it is ready to dispatch transactions. With optimistic data, the socket does not wait, but rather it immediately pretends to the connection manager that the server has established the TCP connection, which causes the connection manager to dispatch the `GET` request immediately. Optimistic data is useful for our defense, as it allows the client to establish a new connection and dispatch the owning transaction at the same time. Optimistic data reduces the number of bursts and thus the amount of padding we need to confuse the attacker.

3.4 Other Implementation Details

Pipelining: Normally, each connection can only handle one HTTP request at a time, and the response should be complete before the connection manager will dispatch another request on the connection. With pipelining, each connection can dispatch multiple HTTP requests at the same time, and the server replies to them one by one, in order. Pipelining may reduce the number of round trips required to load a web page, but the benefit is small and it is currently disabled in major browsers. Additionally, many web servers do not support pipelining [13]. Tor Browser has enabled pipelining as a costless defense against WF [16], but there is no noticeable effect on WF attacks [5, 21]. As the sole intent of using pipelining in Tor is as a WF defense, and it does not benefit `Walkie-Talkie`, we have disabled pipelining.

TLS: We note that even with optimistic data, HTTP requests over TLS are not sent with the connection establishment request. This is because the TLS handshake requires the client to generate a pre-master key from the server's response. The client cannot send the HTTP request without completing the TLS handshake. Therefore, the client will send the HTTP request a short while after a burst has started. Our defense assumes the HTTP request to be at the front of the burst, but our attackers will see their true location; we show this does not weaken our defense in Section 6.

Speculative connections: Firefox may initiate speculative connections if it believes that a transaction for the connection will appear soon. These connections may save a round trip if the speculation is correct, or they may be unused and eventually closed. We have simply disabled speculative connections as they do not obey half-duplex communication and provide no significant benefit to `Walkie-Talkie`.

4. PADDING

To begin, we assume that the client has already taken simple precautions: she is using an encrypted connection to a proxy in order to hide both the content and IP addresses of the sites to which she connects. Further, we assume this encrypted channel pads all packets to a fixed size so as to remove packet size as a possible feature.[3] Then by using half-duplex communication, the client reduces the attacker's possible feature set even further, to simply counts for the number of packets in each burst of traffic. We call the number of packets in each burst the *burst size*, and we call an outgoing burst followed by an incoming burst a *burst pair*. The attacker's information is then limited to a list of pairs of positive integers; e.g. $\langle(2,2),(2,3),(5,20)\rangle$, representing the number of packets in each outgoing burst and its corresponding incoming reply.

The client adds dummy packets that contain no information in order to lower the attacker's maximum possible accuracy. We refer to the attacker who is using the best (information-theoretic) strategy that gives the maximum accuracy as the *perfect attacker*; the perfect attacker's accuracy is the maximum possible attacker accuracy, even if it is unlikely that a real attacker could actually achieve that level of accuracy. The perfect attacker can use all information about the website's true content, the network conditions, the client's defense, and so on, to determine which website(s) could possibly have produced the observed packet sequence. This perfect attacker can only fail, then, when a *collision* occurs; that is, when the same packet sequence (with dummies) could have come from at least two different web pages. In this case, even the perfect attacker cannot correctly determine the web page. We define the *collision rate* as one minus the maximum attacker accuracy. In this section we design a padding defense that protects the client against even a perfect attacker by padding burst traffic sizes; we want to maximize the collision rate while minimizing overhead. We describe how we calculate the collision rate and overhead in Section 5.

We present two variations of a padding defense: a deterministic version, where the defense pads each burst size to a constant, and a random version, where the defense adds a random number of dummy packets to confuse the attacker. The client adds outgoing dummy packets, while a *cooperator* adds incoming dummy packets. The cooperator is located between the attacker and the destination server. In single-hop networks, a cooperating proxy defends against WF attackers eavesdropping on the client's network. In multi-hop networks such as Tor, the cooperator may be any of the hops. An earlier hop saves on bandwidth because padding only exists between the client and the cooperator.[4]

[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.

[3]Note that Tor delivers all data in fixed-size cells.

[4]In both cases proxies that are one hop away from the client (and eavesdroppers on its network) are potential WF attackers, so a

It is not necessary to add enough padding for the perfect attacker's accuracy to be close to zero, because of the base rate fallacy. Due to the low base rate of visits to each single web page, a client protecting her page accesses against an attacker may only need a false positive rate over 5% [18,21] even if the attacker's true positive rate was 100%. This is because the number of false alarms will overwhelm the attacker's classifier.

4.1 Deterministic Padding

For deterministic padding, we perform *rounding* on burst sizes. With Y as a set, we define a rounding function $round_Y(a)$ as follows:

$$round_Y(a) = \begin{cases} \min\{y \in Y | y \geq a\} & \text{if } \exists\, y \in Y : y \geq a \\ a & \text{otherwise.} \end{cases}$$

In our case Y is a small set of positive integers that we call the rounding set. We refer to $cost_Y(a) = round_Y(a) - a$ as the cost of rounding a by the rounding set Y.

For each burst of b packets, the defense pads to $\hat{b} = round_Y(b)$ packets for some rounding set Y by adding dummy packets. We actually have a different rounding set for each burst, but we sometimes elide this detail for simplicity of discussion in what follows. This method of causing collisions is similar to Tamaraw [4]. The overhead of the defense (the number of dummy packets added) is therefore $\sum_b cost_Y(b)$, where b runs across all burst sizes.

We use a deterministic algorithm to find the overhead-optimal rounding set Y given its size $|Y|$. Suppose that the set of integers for which we want to find the optimal Y is a sorted multi-set B. For example, B can be the set of first incoming burst sizes across all observed sequences. Denote $Y_{\leq}(a)$ as $\{y | y \in Y \wedge y \leq a\}$ (and similarly for $Y_{>}(a)$). Then if Y is optimal for B, $Y_{\leq}(y)$ is optimal for $B_{\leq}(y)$ for any $y \in Y$, and $Y_{>}(y)$ is optimal for $B_{>}(y)$ for any $y \in Y$. We can therefore search for the optimal rounding set of size $|Y|$ by enumerating all possible ways to divide B into two contiguous sets, then searching within these two sets for optimal rounding sets of size $|Y|/2$. This is a recursive solution with time complexity $O(|B|^{\log_2 |Y|+1})$. In our implementation we found that the computation cost balloons out of hand at around $|Y| = 10$, but fortunately for our case we do not need such a large Y.

A larger $|Y|$ will lead to a smaller overhead but a lower collision rate. Although we can find the optimal Y given $|Y|$, finding the optimal $|Y|$ for each burst is difficult. In our experiments, we randomize $|Y|$ within $2 \leq |Y| \leq 9$ for each burst and seek to find the optimal solution by evaluating many random solutions.

4.2 Random padding

In this section, we present an alternative scheme that adds a random number of dummy packets each time it is called. The random padding scheme is more complicated because we cannot use the optimization algorithm for deterministic padding. As a result, we must search the solution space of random functions, hoping to find good solutions within a reasonable amount of time.

We denote a probability distribution over the non-negative integers as $X : \mathbb{Z}_{\geq 0} \to [0, 1]$.

Given a packet sequence $s = \langle b_1, b_2, b_3, ..., b_{|s|} \rangle$ with $b_i = (b_{i_{OUT}}, b_{i_{INC}})$, we apply defense D as follows to produce $D(s)$:[5]

1. Padding real burst pairs: From two distributions $X_{i_{OUT}}$ and $X_{i_{INC}}$, we draw $x_{i_{OUT}}$ and $x_{i_{INC}}$ respectively, and add them to b_i, such that $\hat{b_i} = (b_{i_{OUT}} + x_{i_{OUT}}, b_{i_{INC}} + x_{i_{INC}})$. i runs across all burst pairs in the sequence.

2. Adding fake burst pairs: From two distributions $X_{i_{dumOUT}}$ and $X_{i_{dumINC}}$, we draw $x_{i_{dumOUT}}$ and $x_{i_{dumINC}}$, and generate a new fake burst pair $\hat{dum_i} = (x_{i_{dumOUT}}, x_{i_{dumINC}})$. We add fake burst pairs at random with probability p_{dummy} before each real burst pair of packets, which allows multiple fake burst pairs to be added consecutively.

The defense D is therefore defined by the distributions $X_{i_{OUT}}$, $X_{i_{INC}}$, $X_{i_{dumOUT}}$, $X_{i_{dumINC}}$, (for each $i = 1, 2, 3, \ldots$) as well as the probability p_{dummy}. The freedom of choice in these distributions allows our scheme to be tunable (i.e., a client may wish to increase the collision rate by increasing overhead). In the following we describe how we choose these distributions.

4.2.1 Padding real burst pairs

We add a random number of incoming dummy packets to incoming bursts and outgoing dummy packets to outgoing bursts as padding. Padding covers the true number of packets, which the attacker uses as a feature. To implement the addition of dummy packets, both the client and the cooperator send dummy packets when each burst has started. The client detects the start of a new burst from the browser state, and sends dummy packets before sending real packets. The cooperator sees the client's dummy packets, drops them, and similarly starts sending dummy packets before sending real packets.

Parameter selection

We choose the number of packets added to each burst from a uniform distribution over an interval we will determine below. Using the uniform distribution gives no extra information to the attacker except that the true number of real packets is in a range of a particular length, even if the attacker knows an accurate non-uniform prior on the distribution of the number of real packets. We learn the range of the uniform distribution as follows:

- Minimum: The minimum of the range is always zero. If the minimum were, say, $k > 0$, then the attacker could remove k packets from the given stream every time, thus effectively reducing the minimum of the range to 0 for the attacker (but costing the client a larger bandwidth overhead).

- Maximum: We set the maximum L of the range as some proportion r of the mean number of packets in each burst position, taken over all observed packet sequences. We can vary r to change the overhead of the scheme. If the third incoming burst, for example, tends to be large, the third incoming burst of *all* sequences will receive more padding, whereas typically smaller burst positions will receive less padding.

In our experiments we vary r to obtain results with a range of collision rates and overhead.

4.2.2 Adding fake burst pairs

If the number of burst pairs in two packet sequences is different, the defense cannot make them collide by padding real burst pairs as above. We add fake burst pairs to packet sequences to increase the chance of causing a collision. In a fake burst pair, all outgoing and incoming packets are dummy packets; the number of outgoing packets is drawn from X_{dumOUT} and the number of incoming

client using the first hop as a cooperator would not be defending against the first hop itself.

[5]Note that the defense D can be applied with no prior knowledge of the b_i, which is a practical advantage compared to some variants of Supersequence.

packets is drawn from X_{dumINC}. Before each real burst pair, with probability p_{dummy} we insert a fake burst pair. After inserting this fake burst pair, we may insert another with the same probability. Given the number of real and fake burst pairs, any permutation of them has the same occurrence probability. To implement this, the client sends dummy packets without any real data and requests a number of dummy packets in return from the cooperator.

Adding fake bursts anywhere in the packet sequence rather than only at the end is useful for the defense. This is because many packet sequences have multiple bursts with few packets and one or two large bursts with many packets. The position of the large bursts in the sequence is a powerful feature for the attacker; adding fake bursts before or after the large bursts can cover their true location.

Parameter selection

Fake burst sizes should be similar to real burst sizes to maximize the collision rate; we do not want the attacker to be able to distinguish between real bursts and fake bursts with high accuracy. We generate fake burst sizes from distributions fitting the observed burst sizes of real packet sequences (plus the real padding). In Section 6 we will test two simple distributions: uniform and normal distributions. Their simplicity allows us to efficiently estimate the maximum attacker accuracy, as the attacker needs to enumerate over all possible permutations of fake and real bursts. We will also determine if fake bursts help at all (as they cost extra packets) by disabling them and measuring the maximum attacker accuracy.

5. EVALUATION

We will evaluate `Walkie-Talkie` by testing it against known attacks in Section 6.4. However, this is insufficient for arguing that our defense is truly effective against WF. Amongst other shortcomings, this approach does not consider the fact that previous attacks do not expect half-duplex input. If a defense is designed only to defeat known attacks, it is entirely likely that it will soon be defeated by later attacks, a fact that is repeatedly demonstrated in WF [5, 7, 21]. This creates a cat-and-mouse game that cannot produce effective defenses.

Therefore, the mainstay of our evaluation methodology is to demonstrate that our defense is *general*. In other words, we evaluate our defense against a perfect attacker that has maximal accuracy in website fingerprinting. The perfect attacker knows the occurrence likelihood of all possible packet sequences of all pages, and he is always best equipped to classify each sequence to its page. The perfect attacker's classification strategy is simple: it will use a noiseless lookup table matching sequences to web pages. Our choice of a perfect attacker is a continuation of more recent work on general WF defenses [4, 14, 21].

While this information-theoretic attacker has perfected WF, it is not 100% accurate. Specifically, if a packet sequence s can come from two or more possible pages (a collision), the attacker cannot classify s perfectly. Collisions create false positives to deter the attacker: for example, an authority that would take legal action upon observing clients visiting banned pages would be unlikely to act if the false positive rate overwhelms the base incidence rate of clients visiting banned pages. The attacker is not able to affect the base incidence rate of each page. The strategy of the general defense against a perfect attacker is therefore to maximize the collision rate.

5.1 Evaluation of deterministic defenses

Here we show how to calculate the perfect attacker's maximum classification accuracy in the context of collisions caused by a defense D.

The web-browsing client generates a set of packet sequences, denoted as S. Due to half-duplex communication, each sequence $s \in S$ can be written as $s = (b_1, b_2, b_3, ..., b_{|s|})$, where each burst pair b_i is a pair $(b_{i_{OUT}}, b_{i_{INC}})$ of the number of outgoing and incoming packets. For $s, s' \in S$, $s = s'$ if all of their burst pairs are equal. We denote M as the list of all web pages in the data set. Each s has a true class $m \in M$, which the attacker attempts to discern; m may be one of the monitored web pages or the non-monitored class of all remaining web pages.

The attacker has a training set with the correct classification of all elements in S. If the client exposes S to the attacker, the attacker will identify the class of almost all elements, because the collision rate of S is very low. So instead, the client uses a defense that applies some transformation D to each sequence in S by padding (adding dummy packets), such that the attacker only sees $D(S)$. For now we consider only a deterministic D. The attacker is perfect at classification, only failing to classify if there is a *collision*: for $s_1, s_2 \in S$ coming from different classes, $D(s_1) = D(s_2)$. Otherwise, the attacker always succeeds.

The objective of the defense is to maximize collisions. For each $s \in S$, we define the collision set of s as $\{s' \in S | D(s) = D(s')\}$. After applying D, the attacker cannot distinguish between any two sequences in the same collision set. Collision sets impart an equivalence relationship on the elements of S and therefore S is a union of its distinct collision sets. For a collision set $c \subseteq S$, we define n_i as the number of sequences in c that came from class i. To classify any sequence belonging to c, the attacker's optimal strategy (optimizing accuracy) is to guess that all elements in the collision set belong to the class with the highest rate of occurrence in that set. The attacker's overall accuracy over c is therefore $\max_i n_i/|c|$.

5.2 Evaluation of random defenses

When a random padding D is applied to a sequence $s \in S$, the result $D(s)$ is now a probability distribution over possible output sequences. We denote $p_s(\hat{s}) = Pr[D(s) = \hat{s}]$. In this case, we need to calculate the perfect attacker's accuracy differently.

Suppose the attacker sees defended packet sequence $\hat{s}_{real}$, which is drawn from a distribution $D(s_{real})$ (the defense applied on s_{real}). Our perfect attacker knows D, but should not be able to remove dummy packets trivially; that is to say, the attacker cannot extract s_{real} from $\hat{s}_{real}$. Instead, the attacker proceeds as follows. The attacker correctly computes $p_s(\hat{s}_{real})$ for all $s \in S$; for example, the attacker certainly knows that $p_{s_{real}}(\hat{s}_{real}) > 0$. Similar to the deterministic case, the attacker's optimal strategy is to guess the class with the highest likelihood of generating $\hat{s}$.

Denoting S_{real} as the subset of S from the same class as s_{real}, and S_{max} as the most likely class to generate $\hat{s}$ (determined by computing $p_s(\hat{s}_{real})$ for all classes), the chance that the attacker classifying $\hat{s}_{real}$ correctly is 1 if $S_{real} = S_{max}$ and 0 otherwise.

The above is the accuracy of only one defended packet sequence $\hat{s}_{real}$. To evaluate the defense, we apply D multiple times to each element $s \in S$ and perform the calculation as above, as the number of possible outputs of D is too large for computationally exhausting all possibilities. For our experiments in Section 6, we will apply D 10,000 times to obtain small 95% confidence intervals.

5.3 Overhead

We care about two types of overhead: *bandwidth overhead* and *time overhead*. The bandwidth overhead of a defense is the number of dummy packets divided by the number of real packets during page loading. The total number of packets sent across the network is the sum of the numbers of dummy packets and real packets. We write the overhead as a percentage; e.g., a client using a 100% over-

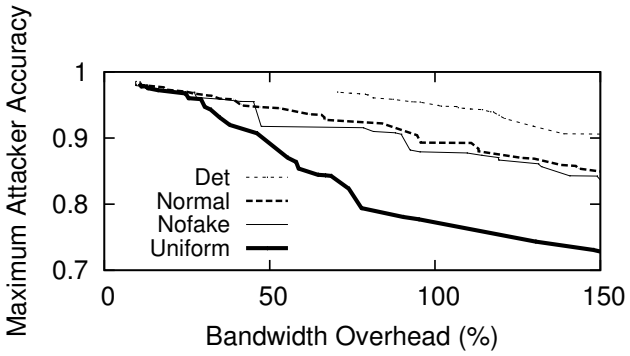

Figure 2: Attacker's accuracy for the four padding algorithms described in Section 6.2. Points are Pareto-optimal in bandwidth overhead and accuracy. Note that the y-axis does not start at 0.

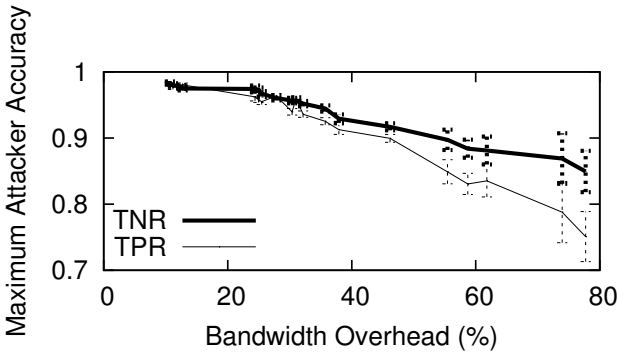

Figure 3: True positive rate and true negative rate with `Uniform` fake bursts. Points are Pareto-optimal in bandwidth overhead and true negative rate. Note that the y-axis does not start at 0.

head defense expects to send twice as many packets as a client who is not using any defense. Our objective is to minimize overhead while maximizing the collision rate. A high overhead implies that the underlying network (proxies or anonymity network) must carry more traffic.

When comparing `Walkie-Talkie` with previous defenses, we will also discuss their time overheads. The time overhead is the amount of extra time needed to load a page, written as a percentage of the amount of time originally needed to load a page. Defenses with constant-rate packet delivery incur a time overhead by forcing packets to wait for the next sending window. For example, Tamaraw [4] intentionally sends data at a low rate in order to decrease the chance of having to send dummy packets, thus lowering bandwidth overhead at the cost of time overhead. Note that we evaluate the time overhead separately from the bandwidth overhead; i.e., the possibility that the extra bandwidth needed will clog the network and lower packet rates does not increase the time overhead. For example, a scheme that simply duplicates each packet in the network has a bandwidth overhead of 100% and a time overhead of 0%.

6. RESULTS

In this section we evaluate `Walkie-Talkie` on data collected from Tor using the methodology in Section 6.1. In Section 6.2 we show that `Walkie-Talkie` is both effective and efficient even against a perfect attacker. In Section 6.3 we compare our defense against known defenses to show the significantly lower overhead of `Walkie-Talkie`. Finally in Section 6.4 we demonstrate that our defense is also highly effective against known successful website fingerprinting attacks.

6.1 Data Collection

We collected our data on Tor Browser 4.5a4 with Tor 0.2.7.0-alpha. We modified Tor Browser to enable half-duplex communication, as described in Section 3. Tor sends data in fixed-size cells, across ephemeral circuits of three hops.

We collected data from Alexa's top pages [1]. We use 100 of the top pages as the monitored set (after removing duplicates due to different localizations or URLs of the same page), and we collected 90 instances of each page in the monitored set. We use the next 10,000 pages in Alexa's top pages as the non-monitored set. We dropped any instance with fewer than 50 cells in it.

We did not use Tor for padding in our experiments; rather, padding is simulated after data collection. This is because we wanted to test out a large number of parameter choices for our padding schemes, and re-collecting data for each set of parameters is infeasible.

6.2 Walkie-Talkie results

In Section 4 we described a deterministic (referred to as `Det` here) and a random scheme for padding in `Walkie-Talkie`. We will test three ways of adding fake bursts in random padding:

1. `Nofake`: No fake bursts are added. Only real bursts are padded.

2. `Normal`: Fake bursts are all normal distributions fitting the observable distribution of real bursts.

3. `Uniform`: Fake bursts are all uniform distributions fitting the observable distribution of real bursts. The range of the uniform distribution is the smallest range that covers 70% of the real burst sizes at the same burst position.

The perfect attacker maximizes his accuracy, which is defined as the proportion of true classifications over the total number of test cases. In our case the number of positive (monitored) test cases and the number of negative (non-monitored) test cases are nearly equal.

To find the optimal parameters, we tested these four algorithms in two steps. First, we obtained 10,000 random selections of parameters (randomization as described in Section 4) for each variation and tested each briefly by obtaining the accuracy for 100 random packet sequences and the overhead for all packet sequences. Obtaining the accuracy is computationally expensive because the attacker needs to evaluate all possible combinations of real and fake bursts. Next, we chose 100 of the best-performing selections of parameters for each variation, and tested them up to 10,000 times against randomly selected packet sequences. This allowed us to obtain 95% confidence intervals assuming a binomial distribution of true and false classifications.

We obtain the Pareto-optimal results (minimal overhead, minimal accuracy) from this list and plot them in Figure 2 for each padding algorithm. We see that `Det`, our deterministic algorithm, did not perform well compared to the random algorithms. At a bandwidth overhead of 80%, the maximum attacker accuracies were respectively 96.0% $\pm$ 0.3%, 91.0% $\pm$ 0.8%, 92.2% $\pm$ 0.5% and 80% $\pm$ 3% respectively for the four algorithms `Det`, `Nofake`, `Normal`, and `Uniform`. Among these algorithms, `Uniform` fake bursts was the most efficient.

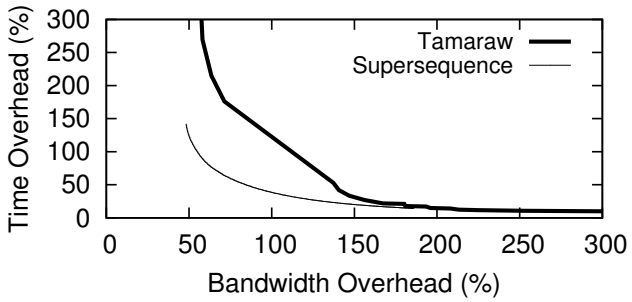

Figure 4: Bandwidth and time overhead tradeoff for Tamaraw and Supersequence morphing. We obtain the tradeoff by varying their cell rates. `Walkie-Talkie` morphing, in contrast, has almost no time overhead and no bandwidth overhead.

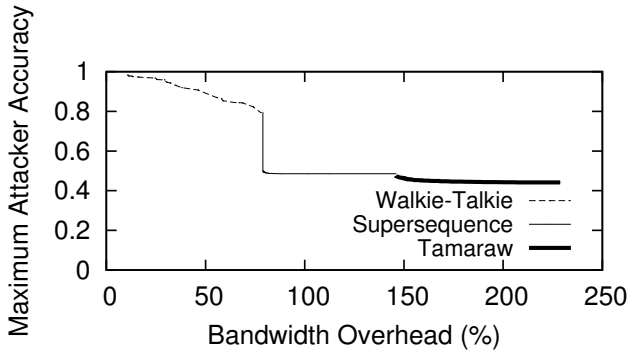

Figure 5: Comparison of maximum attacker accuracy between general defenses. Points are Pareto-optimal in bandwidth overhead and accuracy. With a fixed packet rate, Tamaraw and Supersequence cannot have a lower bandwidth overhead than the region shown in this graph, and they have a large time overhead between 50% and 150%. Walkie-Talkie has almost no time overhead.

The accuracy as obtained above is an average of the true positive rate and true negative rate. We are also interested in observing how each rate varies individually because a low true negative rate (high false positive rate) would severely hamper the attacker in our low base rate scenario. If the attacker categorized a monitored page as a different monitored page, we count it as a false positive. To obtain the true positive rate and true negative rate, we tested `Uniform` fake bursts specifically for the open-world scenario, and show the results in Figure 3. In this figure, we plot the Pareto-optimal points in terms of bandwidth overhead and true negative rate. Nevertheless the graph shows that the true positive rate drops faster than the true negative rate with increasing overhead. At an overhead of 35%, the false positive rate exceeds 5%; at an overhead of 55%, the false positive rate exceeds 10%. We therefore see that `Walkie-Talkie` is able to cause significant confusion to even a perfect attacker with relatively small overhead, especially if a low false positive rate is sufficient (as is the case if the base rate of visiting the monitored pages is low). In the next section we will show that this overhead is much smaller than that of previous defenses.

6.3 Comparison with previous defenses

We compare our defense with previous known defenses to show the advantages of `Walkie-Talkie`. Researchers have published two general defenses: Tamaraw [4] and Supersequence [21]. Both of them use constant-rate packet delivery and rounding the total number of packets to a rounding set in order to produce collisions.

We divide each defense into two components: morphing and padding, and compare them on each component.

1. Morphing: The defense restricts the feature set available to the attacker by morphing packet sequences into another format. `Walkie-Talkie` uses half-duplex communication to morph, whereas Tamaraw and Supersequence use constant-rate packet delivery. Morphing in `Walkie-Talkie` has almost no overhead and morphing in the other defenses has a large overhead.

2. Padding: The defense adds dummy packets to cause collisions. For `Walkie-Talkie` we use `Uniform` fake bursts as described in Section 6.2. Tamaraw and Supersequence use rounding sets to add dummy packets.

Morphing: For `Walkie-Talkie`, morphing with half-duplex communication incurs no bandwidth overhead but some time overhead waiting between bursts. The mean amount of time taken to load each page in our data set was 17.8 ± 6.6 seconds with half-duplex communication compared to 16.4 ± 6.0 without it. The large variance is caused by fluctuating network conditions in Tor. The time overhead at the mean is 8.5%. `Walkie-Talkie` incurs a low time overhead because the number of round trips required to load the page remains the same with and without half-duplex communication, and the round-trip time dominates page load time in multi-hop anonymity networks such as Tor.

For Tamaraw and Supersequence, morphing with constant-rate packet delivery is much more expensive. The cell rate in Tamaraw and Supersequence determine the time overhead and bandwidth overhead: a lower cell rate sends real packets more slowly, potentially decreasing the necessity for sending dummy packets, and so decreasing the bandwidth overhead; however, the lower cell rate increases the time overhead. We show the tradeoff between time overhead and bandwidth overhead of Tamaraw and Supersequence in Figure 4 by varying the cell rate.[6] From the plot we can see that if we want to implement Tamaraw or Supersequence with a reasonably low time overhead, their bandwidth overhead must be very high. For example, to reach a time overhead of 20%, the bandwidth overhead would be at least 180% and 150% respectively even at no collision rate.

Realistically, Tamaraw and Supersequence are likely to have an even higher overhead because the optimal cell rate depends on the parameters of the connection itself, which may be unpredictable to the client, whereas for `Walkie-Talkie` we do not limit the packet rate. This shows us that half-duplex communication is much more realistic for a low-latency web-browsing client than constant-rate packet delivery.

Padding: Next, we evaluate the padding components for the general defenses. To do so, we measure the perfect attacker's maximum accuracy versus bandwidth overhead. Relative to the other defenses, padding is harder for `Walkie-Talkie` because its morphing component left more features for the attacker. For Tamaraw and Supersequence, we select the cell rate that minimizes the bandwidth overhead where the time overhead is 50%. At this level,

[6]Supersequence originally did not have a parameter to control the cell rate; we added it to the algorithm.

the bandwidth overhead for Supersequence is at least 80% and the bandwidth overhead for Tamaraw is at least 140% from constant-rate packet delivery.

We plot the Pareto-optimal results for the three defenses in Figure 5. We see that while Tamaraw and Supersequence perform better than `Walkie-Talkie` in their respective ranges of bandwidth overhead, their minimal bandwidth overhead is much higher from morphing. Furthermore their accuracy is not tunable when the cell rate is fixed. For `Walkie-Talkie`, between bandwidth overheads of 25% to 50% the perfect attacker's false positive rate ranges between 4% and 10%, which is often more than sufficient to overwhelm the base rate of page visits in the sensitive data set.

Note that for Tamaraw and Supersequence, padding further increases the time overhead, such that the points in Figure 5 have a time overhead between 50% and 150% due to constant-rate packet delivery forcing padding packets to wait. For `Walkie-Talkie`, padding real bursts does not add to the time overhead, but adding fake bursts does. The extra time overhead depends on the network; in the worst case, where page load time is dominated entirely by the round-trip time, the extra overhead is between 8% and 15% for our best parameters.

We see that morphing with constant-rate packet delivery causes Tamaraw and Supersequence to be overkill against an open-world attacker. `Walkie-Talkie` fills an important niche: a low-cost algorithm that deters even a perfect attacker by imposing a nontrivial false positive rate.

6.4 Performance against known attacks

We have shown that our defense performs well against a perfect attacker in Section 6.2. It is also interesting for us to measure the effectiveness of `Walkie-Talkie` against known attacks. We implemented eight known WF attacks and tested each of them against `Walkie-Talkie` using `Uniform` fake bursts. 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 40 instances of each of the 100 monitored pages for training and testing with 10-fold cross validation.

We show the results in Table 1 under four columns: the original reported accuracy by the authors of the attack (Reported), the accuracy after morphing (half-duplex communication) but no padding (NoPad), the accuracy with light padding (LightPad) and the accuracy with heavy padding (HeavyPad). The details of the LightPad and HeavyPad settings are in the caption to the table.

We can see from Table 1 that none of the attacks comes close to the perfect attacker's maximum accuracy. Jaccard and MNBayes are highly inaccurate in our case because they rely on unique packet lengths, and there are no unique packet lengths in our scenario. Out of all the attacks, SVM by Panchenko et al. [15] appears to suffer least from the padding. Indeed, previous authors [4, 7] have noted the resilience of this attack against random noise, possibly because its use of a "kernel trick" transforming distances between packet sequences allows greater flexibility in ignoring dummy packets. Later attackers using the SVM and kNN did not use this kernel trick. We note that the comparison between the original reported accuracy and the accuracy under morphing (first and second columns) may be due to half-duplex communication restricting the feature set available to the attacker, but it may also be because those experiments had different experimental setups. The last four attacks in the table use the same setup as ours.

Table 1: Accuracy of known attacks against `Walkie-Talkie`. The accuracy reported by the original authors (with no defense) is included for comparison. The NoPad column is only morphing with no noise. LightPad includes morphing and padding using `Uniform` fake bursts with 23% bandwidth overhead and maximum attacker accuracy of 0.97. HeavyPad includes morphing and padding using 64% bandwidth overhead and maximum attacker accuracy of 0.84.

Attack	Reported	NoPad	LightPad	HeavyPad
Jaccard [11]	0.72	0.01	0.01	0.01
Naive Bayes [11]	0.68	0.44	0.28	0.17
MNBayes [9]	0.96	0.06	0.06	0.04
SVM [15]	0.73	0.65	0.51	0.36
DLevenshtein [5]	0.87	0.55	0.27	0.14
OSAD [22]	0.91	0.68	0.48	0.26
FLevenshtein [22]	0.70	0.51	0.34	0.19
kNN [21]	0.91	0.70	0.38	0.11

7. ALTERNATIVE DEFENSES

In Section 1, we listed three design goals of `Walkie-Talkie`. In this section, we present three variants of `Walkie-Talkie` that do not meet each one of the three goals. Each variant has a significantly lower overhead and maximum attacker accuracy. A summary is given as follows:

1. Without generality: Rather than measuring our defense effectiveness against a perfect attacker, we investigate what overhead our defense needs in order to defeat the state-of-the-art kNN attack by Wang et al. [21]

2. Without ease of use: We show that the client can use a nearly costless defense with a different objective: rather than protecting all page accesses, the client chooses a set of web page accesses she wishes to conceal and only adds noise to them, attempting to mimic a set of frequently visited web pages. Since the client needs to configure the sensitive and popular pages, this variant requires some effort from the client.

3. Without decentralization: We show that by using a frequently updated central database, which stores some packet information about web pages, the client can add padding in a much more efficient manner.

We present each variant and briefly evaluate its effectiveness in the following sections.

7.1 Without generality

One of our design goals is that the defense should be general; i.e., even a perfect attacker should not be able to decide which packet sequences come from monitored web pages under the defense. This is a strong requirement: for example, many web servers contain advertising, varying or personalized content, which may be unpredictable to any reasonable attacker, but the perfect attacker predicts such content perfectly. We investigate what overhead we would need if we wanted to defeat only the best known attack, which is currently the kNN attack by Wang et al. [21], rather than all possible attacks with all possible auxiliary information available to the attacker. Their attack is a good candidate for our case because it is designed to defeat defenses by using feature extraction to exploit limited defenses that left some features unprotected.

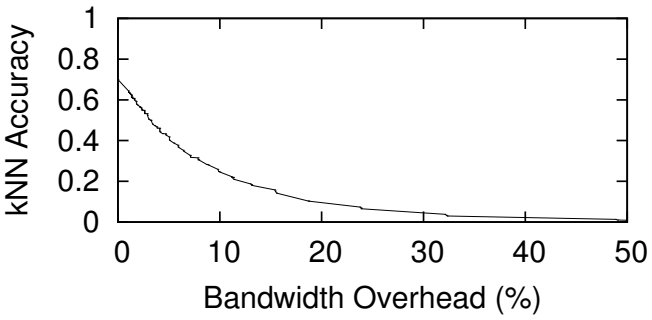

Figure 6: kNN [21] accuracy versus overhead with the burst-splitting defense described in Section 7.1.

Wang et al. suggest that gathering features about contiguous sequences of packets in the same direction is effective for their attack. Therefore we consider a simple burst-splitting defense: the burst-splitting defense adds random incoming and outgoing dummy packets to break contiguous sequences, on top of half-duplex communication. We then test the kNN attack against the burst-splitting defense, varying the total number of dummy packets, to obtain a tradeoff between bandwidth overhead and accuracy. We show the results in Figure 6. We can see that the attack accuracy drops sharply with only a small amount of noise added. In comparison, recall that the attacker's maximum accuracy was around 96% when we added 25% overhead to `Walkie-Talkie`. For the burst-splitting defense, kNN attack accuracy drops to 6% with the same amount of overhead. At around 50% overhead, the accuracy is 1%, the random guessing accuracy.

Although the variant we describe here is highly successful with low overhead, it would perform very poorly against a perfect attacker. In general we believe that designing a defense that specifically defeats the state-of-the-art attack is a flawed approach. In practice, the attacker may store the client's packet sequences, taking his time to figure out the defense and tune his attack until he succeeds with high confidence. What this variant does is the reverse: we tuned the defense to defeat the attack. A realistic client would not be able to tune her defense against a known attack; the client must choose a good defense with no knowledge of how it will be attacked.

7.2 Without ease of use

We designed our defense so that it would require no input from the client; in fact, it is invisible to the client. This is because we believe privacy solutions should be easily usable, and the popularity of Tor is evidence of this: thanks to the Tor Browser, using Tor to browse the Internet is as easy as using a common browser to do so. However, if the client is willing to put in some effort for a more efficient `Walkie-Talkie`, she may use the variant described in this section.

Using this defense, the client does not protect all page visits from the WF attacker. Rather, the client selects a small set of pages, the *sensitive set* (a set of infrequently accessed pages). When the client visits a sensitive page, the defense adds noise, such that it looks like a page from the *popular set* (a set of frequently accessed pages). The popular set could be Alexa's top 100 pages. No noise is added to any other page. Given that the base rate of accessing the sensitive set is very low, the perfect attacker's accuracy and the overhead of the scheme are both close to 0, the exact value depending on the base rate of visiting pages from the sensitive set.

Mimicry may fail if the number of real packets when visiting a sensitive page exceeds the number of packets we wanted to add. Failing to mimic a page does not completely reveal the original page; however, the attacker may guess that mimicry is happening and gain some information about the true packet sequence. Therefore we only need to ensure that the failure rate is low.

We test the failure rate using the following algorithm. Our data set is divided into training instances and testing instances of each page. Using the training instance, the client computes a characteristic sequence for each sensitive page, which is a supersequence of most training instance sequences. The client then checks the packet sequences of all popular pages: if the popular packet sequence is a supersequence of the characteristic sequence, the client attempts to mimic that packet sequence. Mimicry fails if the testing packet sequence (which the client did not train on) is *not* a subsequence of the mimicked packet sequence. We randomized the sensitive set by picking 10 random pages in Alexa's top 100 pages, and the popular set as the other 90 web pages.

We found a mean failure rate of $4\% \pm 3\%$, and each page had a mean of 95 packet sequences that it could mimic. The client can lower the failure rate further by only attempting to mimic very large packet sequences.

7.3 Without decentralization

Using `Walkie-Talkie`, the client treats each page the same: the noise is randomly chosen the same way for each page. However, if the client were to know the packet sequence of the page in advance, then the client could pad each sequence differently to minimize bandwidth overhead. However, such knowledge requires the use of a centrally maintained database of packet sequences. The database owner must frequently visit a large set of pages to gather information on their packet sequences in order to keep the database up to date. Clients can query the database using private information retrieval. While expensive to maintain, this variant can significantly save on the client's bandwidth overhead.

This setting is similar to the class-level knowledge setting investigated by Wang et al. [21], and they have proposed an algorithm to optimize the defense. In this work we test a variation of their algorithm in depth. The database gathers packet sequences of all pages, and calculates a characteristic sequence for each page, which is a supersequence of most of its packet sequences. Then, the database merges characteristic sequences: it takes the characteristic sequences of two different pages, calculates their supersequence, and assigns the new supersequence as the characteristic sequences of both pages. The database repeats this process to increase the collision rate. The client will attempt to load each page by padding up to its characteristic sequence.

We test the above algorithm by varying the computation of the characteristic sequence as well as the number of merges. We show the Pareto-optimal results (in bandwidth overhead and maximum attacker accuracy) in Figure 7. With only a 5% overhead, the attacker's maximum accuracy drops below 90%; 25% overhead is enough for the attacker's maximum accuracy to drop below 80%. However, there is a caveat: some pages remain completely undefended as merging their characteristic sequence with any other would greatly increase the overhead. A defense that defends all pages in the database would have a much higher overhead.

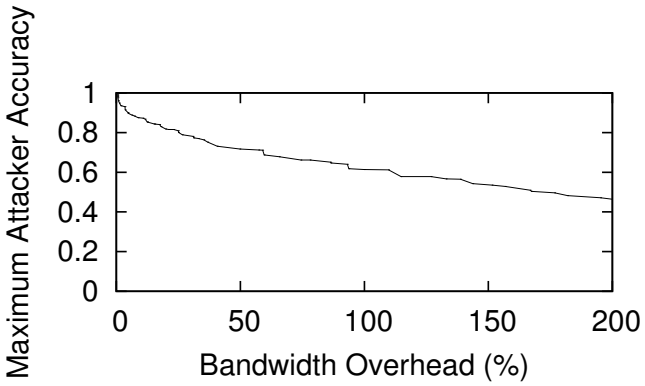

Figure 7: Best results for maximum attacker accuracy versus overhead when the client queries a central database before packet padding, for Alexa's top 100 sites.

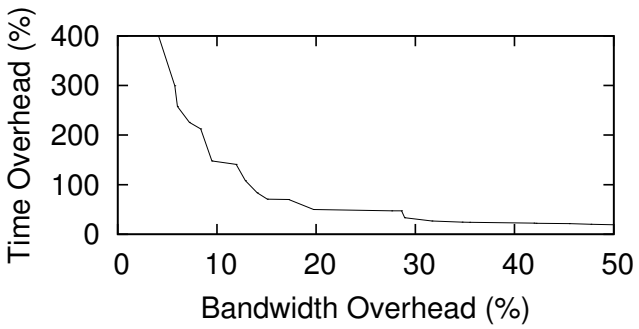

Figure 8: Bandwidth and time overhead tradeoff if we implemented constant-rate packet delivery on top of `Walkie-Talkie` to remove timing as a metric for the attacker.

8. DISCUSSION

8.1 Reproducibility of our results

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, our deterministic and random padding algorithms, our Pareto-optimal selections of parameters, and the three alternative defenses.
- 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 is available at the CrySP site `https://crysp.uwaterloo.ca/software/webfingerprint/`.

8.2 Timing

In this work we made the simplifying assumption that the attacker does not use precise packet timings as a WF attack feature. While this does limit the power of the supposedly perfect attacker, there are several reasons for this assumption:

1. Timing is rarely used as a WF attack metric. None of the attacks tested in this work, which were each the state of the art at the time, used timing as a metric. We are aware of only one attack that uses timing: an attack by Bissias et al. [2] (2005), which is not accurate by state-of-the-art standards. Improved attacks have not found timing to be a powerful metric.
2. On an anonymity network, a single client must switch regularly between relays to protect her identity. On Tor, the client switches circuits every 10 minutes (by default). This means that network conditions, specifically the capacity of the network as experienced by the client, fluctuates very quickly relative to the training and testing time of the attacker. We can therefore expect that there is a significant disparity between the client's timing and the timing in the attacker's training database.
3. Specifically for Tor, the bottleneck in the client's connection to a web server is almost always at the connections between Tor relays. [19] Interpacket timing would then only reveal the packet rate induced by the Tor network.
4. For a mass surveillance operator that needs to attack a large number of clients at once, each with their own network conditions, the attacker needs to adjust the classifier for each set of network conditions. In the worst case, the attacker would need a different data set and classifier for each client. Essentially, the assumption that timing is not a usable metric can be understood as asserting that the attacker does not have enough computing power to store, train, and update such a classifier for each client.

Nevertheless, we can remove the assumption that the attacker does not use timing as a feature with a slight cost to the bandwidth and time overhead, simply by enforcing a constant packet rate in half-duplex communication. When each side begins sending its burst of data, it does so at a constant rate while sending dummy packets if there is no real data. After sending several dummy packets, the sender recognizes that the burst has ended, and stops sending data until the other side finishes their burst. This approach has two further advantages: the defense can be implemented purely at the socket level (only modifying Tor, not the browser), and it enables rounding as a defense. Note that unlike Tamaraw and Supersequence, each side in our approach stops talking between bursts, allowing the packet rate to be high, saving significantly on overhead. On the other hand, this approach does carry the implementation problems of constant-rate packet delivery as addressed by Cai et al. [3], as do Tamaraw and Supersequence.

We briefly test this approach on our Tor data set, varying the cell rate as well as the number of consecutive dummy cells before each side recognizes that the burst has ended. If the burst ends early (before all real cells have been sent), we count the rest of the real cells as errors. In the implementation, those cells would be sent in the next burst. We discarded any combination of cell rate and consecutive dummy cells where the error rate was more than 5%. We show the results in Figure 8. We can see that there are relatively efficient choices of parameters. For example, to achieve a bandwidth overhead and time overhead of 20% and 48% respectively, we send a fixed rate of 250 outgoing cells per second, 73 incoming cells per second, terminating each burst at 37 consecutive dummy cells outgoing and 37 consecutive dummy cells incoming. Decreasing the bandwidth overhead further would greatly increase the time overhead (similar to Tamaraw and Supersequence).

9. 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` uses half-duplex communication to limit the attacker's feature set and random padding to confuse even a perfect attacker. Given that website fingerprinting is performed in the low base rate scenario, we find that previous general defenses such as Tamaraw and Supersequence are overkill: they incur a large bandwidth and time overhead because of constant-rate packet delivery, whereas `Walkie-Talkie` can cause a sufficient false positive rate efficiently. Our alternative defenses showed that the defense is even more efficient under less restrictive design goals.

10. 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, 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.

[4] 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.

[5] 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.

[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] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.