Website Fingerprinting: Attacks and Defenses

by

Tao Wang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2015

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Website fingerprinting attacks allow a local, passive eavesdropper to determine a client's web activity by leveraging features from her packet sequence. These attacks break the privacy expected by users of privacy technologies, including low-latency anonymity networks such as proxies, VPNs, or Tor. As a discipline, website fingerprinting is an application of machine learning techniques to the diverse field of privacy.

To perform a website fingerprinting attack, the eavesdropping attacker passively records the time, direction, and size of the client's packets. Then, he uses a machine learning algorithm to classify the packet sequence so as to determine the web page it came from. In this work we construct and evaluate three new website fingerprinting attacks: `Wa-OSAD`, an attack using a modified edit distance as the kernel of a Support Vector Machine, achieving greater accuracy than attacks before it; `Wa-FLev`, an attack that quickly approximates an edit distance computation, allowing a low-resource attacker to deanonymize many clients at once; and `Wa-kNN`, the current state-of-the-art attack, which is effective and fast, with a very low false positive rate in the open-world scenario.

While our new attacks perform well in theoretical scenarios, there are significant differences between the situation in the wild and in the laboratory. Specifically, we tackle concerns regarding the freshness of the training set, splitting packet sequences so that each part corresponds to one web page access (for easy classification), and removing misleading noise from the packet sequence.

To defend ourselves against such attacks, we need defenses that are both efficient and provable. We rigorously define and motivate the notion of a provable defense in this work, and we present three new provable defenses: Tamaraw, which is a relatively efficient way to flood the channel with fixed-rate packet scheduling; Supersequence, which uses smallest common supersequences to save on bandwidth overhead; and Walkie-Talkie, which uses half-duplex communication to significantly reduce both bandwidth and time overhead, allowing a truly efficient yet provable defense.

Acknowledgements

I would like to thank all the great teachers in my life who have inspired, educated, and supported me, and to whom I am and will always be thankful. I owe my accomplishments to these great teachers, and without them this work would not have been possible.

First, my father, Susheng Wang, and my mother, Wei Yang, great teachers in their own right, but especially so to me. They were responsible for my growth both physically and mentally. They have always shown encouraging pride in the little odd things I did as a child, which has led to this dissertation.

Next, to those who have taught me not only in spirit, but also in school. A full list would be far too long, so I can only include a select few who were especially invested and influential in my education: Mr. Kwan Yau Wong, who taught me Chinese; Mr. Kip To Yip, who taught me violin; Mrs. Mei Fong Chow Yu, who taught me math; Dr. Zhiyu Yang, who taught me physics; and Dr. Ian Goldberg, who taught me security and privacy, and who as my supervisor taught me the many skills necessary to survive as a graduate student.

I would also like to thank my friends and peers, who have not only supported me, but taught me so much of what I know. Of my high school friends, Mark Tam taught me math; Ching Cheng taught me philosophy and language; Leo Tang taught me physics and astronomy; and Lincoln Hui taught me English. I have always admired these great friends of mine. I am also especially grateful to members of the Cryptography, Security, and Privacy (CrySP) laboratory at the University of Waterloo, who have supported and taught me so much. These include, but are not limited to, Ryan Henry, Tariq Elahi, Cecylia Bocovich, and Nik Unger. I am proud to call them my friends.

For their generous financial support throughout my PhD studies, I would like to thank the Ontario government and participating instutitions for a Ontario Graduate Scholarship, the University of Waterloo for a President's Graduate Scholarship, and the David R. Cheriton School of Computer Science for a Cheriton Graduate Scholarship.

This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca) and Compute/Calcul Canada.

To Wei Yang and Susheng Wang;
and to all whom I never want to disappoint.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Website Fingerprinting

Everyone has a right to privacy. Warren and Brandeis, in their famous law review article "The Right to Privacy", describe it as a natural extension of the principle that "the individual shall have full protection in person and in property" [WB90]. They observe that:

> The common law secures to each individual the right of determining, ordinarily, to what extent his thoughts, sentiments, and emotions shall be communicated to others. ... The existence of this right does not depend upon the particular method of expression adopted.

As the Internet grows in size and impact in the global economy, privacy on the web has become a growing concern [GWB97, Gol03, Gol07], and web-browsing behaviour has become increasingly valuable. Academics [Gol07, Lab07], technologists [DMS04, Psi15], civil liberties organizations [COR15], and legislators [Uni13] are actively presenting new solutions to protect user privacy.

Privacy can be achieved by dissociating identity from activity. Under the framework of web browsing, the identity is the client's IP address, and the activity is the web page she is browsing (her destination). Our client of interest is a privacy-conscious web browser who uses both encryption and proxies to protect her privacy: she separates information about her destination from information about her location. Without encryption, the attacker may read her packet content to identify the web page. Without a proxy, her packet headers will contain the true destination.

This thesis examines a threat to web-browsing privacy, **website fingerprinting** (WF): website fingerprinting is a class of attacks that allows local, passive network eavesdroppers to identify the web page accessed by a client, using traffic analysis.

The local, passive network eavesdropper (the attacker) is a powerful but invisible threat. The attacker starts with knowledge of the web-browsing client's identity, and collects the client's observable traffic as a *packet sequence*. Applying machine classification to the client's packet sequence, the eavesdropper guesses which web page she is visiting. Thus, the eavesdropper re-associates the client's identity with her activity, despite her efforts to separate the two, compromising the client's privacy. Website fingerprinting exposes the gap between security and privacy: the attacker can identify the true destination of a web-browsing client (and thus her behaviour) without decrypting any of her packets.

One way to browse the web using proxies with encryption is to use low-latency anonymity networks such as Tor. Tor is one of the most popular privacy-enhancing technologies, used by around two million users every day [Tor15]. Tor consists of volunteer nodes that relay encrypted traffic between the client and the web server, ensuring that no single party is able to know both the client's source and her destination. If website fingerprinting is indeed an accurate and realistic threat, as we will endeavour to show in this thesis, then anonymity networks such as Tor will have failed to protect user privacy; in fact, they may exacerbate the attack by allowing untrusted volunteer nodes to have access to client traffic.

Tor implements a defense against website fingerprinting, altering the packet sequence to cover some of its identifiable features. Indeed, some previous work on website fingerprinting has failed against Tor [HWF09]. Tor actively responds to the research community: in response to website fingerprinting attacks that succeeded against Tor, Tor has changed the specifics of its defense [Per13]. There are many recent works investigating website fingerprinting on Tor [PNZE11, CZJJ12, DCRS12, JAA+14]. This thesis examines website fingerprinting attacks and defenses for privacy enhancing technologies in general; due to Tor's real-world impact and active response to research, we use Tor as the chief example to present our results.

Website fingerprinting is a relatively new threat to web privacy; the first successful attack on Tor in the closed-world scenario was published in 2012 by Panchenko et al. [PNZE11], and we showed success on Tor in the open-world scenario as well in 2014 [WCN+14]. For this reason, and also because website fingerprinting is inherently invisible to the victim, there is little evidence of website fingerprinting attacks in the wild. Nevertheless, similar traffic analysis attacks for much simpler goals (e.g. detecting any traffic from a specific application) are present in large-scale projects such as XKEYSCORE [Moo14] and the Golden Shield Project [EZ05].

1.2 Thesis statement

If website fingerprinting were truly practical as an attack, it would be a significant threat to many privacy-enhancing technologies that seek to protect the web-browsing client against malicious eavesdroppers. This work offers state-of-the-art techniques that push the boundaries on both website fingerprinting attacks and defenses. In doing so, we endeavour to establish the following thesis statement:

> We can effectively and practically compromise an undefended web-browsing client's privacy with website fingerprinting attacks, and we can also effectively and practically defend against all possible website fingerprinting attacks.

On both the offensive and defensive sides, we claim to achieve effectiveness and practicality. We expand on each claim as follows:

1. Effective attacks: We design new attacks that achieve a significantly higher true positive rate and lower false positive rate compared to the best previous attacks. This allows us to accurately distinguish between a large set of web pages even in the open-world scenario, where there is no limit to the set of web pages the client can visit.

2. Practical attacks: Our state-of-the-art attack is fast enough that one machine can classify hundreds of packet sequences per second, thus potentially deanonymizing large numbers of anonymity network clients. We bridge the gap between laboratory conditions and realistic conditions by presenting methods that allow attacks to operate well on packet sequences collected in the wild.

3. Effective defenses: We define the notion of a general defense that succeeds in thwarting any possible website fingerprinting attack by colliding packet sequences. We design several effective general defenses, allowing a client to be sure that she can resist website fingerprinting.

4. Practical defenses: Some website fingerprinting defenses are impractical, as they carry a large bandwidth and time overhead. We design and implement a state-of-the-art defense that uses half-duplex communication while browsing to collide packet sequences efficiently. This scheme has a highly tunable overhead, which can be small enough for the defense to be applied en masse to privacy technology users (e.g. Tor clients).

1.3 My Work

This thesis describes and analyzes work on website fingerprinting attacks and defenses from many researchers. In this section, I list the work I have done in this field so that there is no confusion between my work and that of other researchers.

1. "Improved Website Fingerprinting on Tor" [WG13b]. In this work we describe two new WF attacks based on an attack by Cai et al [CZJJ12]. We call our new attacks `Wa-OSAD` and `Wa-FLev`. These attacks respectively improve classification accuracy and cut down training/testing time. We developed a new data collection methodology to collect data in this work (Section 2.4), so that our experimental data is more representative of real clients. This work was published at WPES 2013.

2. "Efficient Attacks and Provable Defenses for Website Fingerprinting" [WCN+14]. In this work we describe a new attack called `Wa-kNN` that significantly improves both classification accuracy and training/testing time compared to previous attacks (Section 3.4.3) by using a k-Nearest Neighbours classifier. We also introduce the notion of a general defense called Supersequence, against which no website fingerprinting attacker can succeed. This work was published at USENIX Security 2014.

3. "Comparing Website Fingerprinting Attacks and Defenses" [WG13a]. In this technical report we develop a new methodology to compare website fingerprinting attacks by analyzing their features sets. We also describe Tamaraw, another general defense with good properties. Cai et al. published a work with us at CCS 2014, "A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses" [CNW+14], which included a portion of the technical report, including Tamaraw, as well as work I am not claiming as my own. To clarify the distinction between my work and theirs in this thesis, we will refer to results in the technical report as our work, and other results in the CCS paper as work by Cai et al.

4. "On Realistically Attacking Tor with Website Fingerprinting" [WG15a]. In this work we tackle several issues with previous website fingerprinting works that impede its realistic implementation. To that end, we develop algorithmic methods to bridge the gap between theory and practice for website fingerprinting. This work is currently under submission to PoPETs 2016.

5. "Walkie-Talkie: An Effective and Efficient Defense against Website Fingerprinting" [WG15b]. We develop a new, tunable general defense, Walkie-Talkie, with significantly lower time and bandwidth overhead than previous general defenses, and we show that this

defense succeeds at defending web-browsing clients against even a perfect attacker. This defense is practical and we have implemented it for Tor. This work is currently under submission to NDSS 2016.

Distilling the above, we developed the new attacks labeled as `Wa-OSAD`, `Wa-FLev` and `Wa-kNN`, and the new defenses labeled as Tamaraw, Supersequence and Walkie-Talkie. Other attacks and defenses are not our work; we include them for comparison. This thesis also presents a new systematization and comparison of all these works.

Chapter 2

Background

Website fingerprinting (WF) is an application of machine learning techniques to attack web privacy. We discuss web browsing and website fingerprinting in Section 2.1, and we give the background of machine learning in Section 2.2. We proceed to give a list of terminology in Section 2.3, and in Section 2.4 we describe the data sets we used to present our results as well as our data collection methodology.

2.1 Web Browsing

In this section we describe the mechanics behind web browsing. We divide our discussion in two sections: we describe regular web browsing in Section 2.1.1, and web browsing with Tor in Section 2.1.2.

2.1.1 Web Browsing without Tor

We examine how browsers load web pages 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 [FR14].

A web page comprises a number of resources, such as text, images, and code. Loading the web page involves loading all of its resources. Internally to the browser, the process of requesting and loading a resource (a GET request) is called a transaction. Transactions also include other requests that may modify the server such as POST requests.

Transactions start with an HTTP request from the client. HTTP is an application-layer protocol under TCP/IP that serves as the backbone of web browsing. HTTP attempts to adjust for the shortcomings of TCP/IP by minimizing the number of TCP connections necessary to load the page. At the time of writing of this thesis, HTTP 1.1 is the most recent version of HTTP, while HTTP 2.0 is under discussion and implementation. Under HTTP 1.1, a single TCP connection can only handle a single transaction at a time.[1] HTTP Pipelining creates a minor exception to this rule, where the client can make several GET requests in a queue, and the server will respond to them one-by-one in order.[2]

To dispatch transactions, the browser creates or re-uses long-lived TCP 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.

Browsers have a limit on the number of connections per server or in total. This number varies across browsers and browser versions; as an example, Firefox 38.0 allows 6 connections to each server and 256 connections in total.[3] If the number of transactions has reached the limit, the browser stores new transactions 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.

2.1.2 Web Browsing with Tor

Tor is a popular low-latency anonymity network supported by volunteer nodes that relay traffic for Tor clients. Tor clients construct a circuit of three nodes (the entry guard, the middle node, and the exit node) and use multi-layered encryption; the exit node is able to see the original TCP data sent by the client, and the entry node knows the client's identity. Without collusion between entry and exit nodes of the same circuit, none of the relays (or eavesdroppers on those relays' networks) should be able to link the client's identity with her destination.

Tor performs no mixing to keep latency minimal, which renders it susceptible to timing attacks. While it is well known that global adversaries controlling both ends of the client's circuit

[1] HTTP 2.0 allows servers to handle several transactions at the same time in one TCP connection.

[2] Currently, pipelining is not enabled by default in Internet Explorer, Firefox, or Chrome, and it is not supported by many servers [LZC+11].

[3] It is interesting to know that many CDNs work around the low number of connections per server by distributing web page resources across different servers to increase the number of connections and thus lower the time-to-last-byte.

can break Tor's anonymity guarantees using a timing attack, it is assumed that such adversaries are rare due to Tor's global nature [DMS04]. However, website fingerprinting assumes a much weaker adversary: a local, passive adversary that only observes the client's end of the connection, such as the client's ISP, a packet-sniffing eavesdropper, wiretapper, legally coercive forces, or any Tor entry node itself.

Each circuit multiplexes multiple *streams* corresponding to different TCP connections. Tor nodes cannot distinguish between Tor streams, and our attacker will not be able to do so as well. By default, Tor circuits last for 10 minutes, after which they are marked as dirty, and Tor will not assign new connections to dirty circuits.

Tor sends data in fixed-size (512-byte) Tor cells. If there is not enough data to send, Tor pads cells with encrypted zeroes. Since every cell is encrypted, the website fingerprinting attacker cannot tell the true length and content of each Tor cell. Tor packages its cells into TLS records, which the network then splits into TCP segments. To derive cell sequences from the observed TCP segments, we merge TCP segments into their original TLS records, and then divide by 512 to find the number of cells. We use cell sequences to represent Tor data rather than packet sequences.

Tor also uses cells for other purposes, such as circuit construction, circuit destruction, and flow control. For flow control, Tor uses SENDME cells, which Tor sends automatically once per 50 incoming cells for each stream and per 100 incoming cells for each circuit. As SENDME cells are noise, and they interrupt bursts of traffic (bursts are useful for website fingerprinting; for example, see Section 3.1.9), we will describe an algorithm to try and remove SENDME cells in Section 4.5.2.

To browse the web, Tor recommends that users access Tor through Tor Browser. Tor Browser is a modification of Firefox; it aims to protect client privacy against a number of adversaries. It does not keep any information on the hard disk, so that an attacker who can seize the hard disk cannot identify user behaviour. It disables Flash by default, and uses HTTPS Everywhere to try to access every page with encryption.

Tor Browser also attempts to behave the same way for every client to prevent browser fingerprinting [AJN+13], which in a sense is the dual problem of website fingerprinting: while the website fingerprinting attacker is local to the client and attempts to identify her behaviour, the browser fingerprinting attacker is located at the server and attempts to identify clients he has seen before. The consistent behaviour of Tor Browser across operating systems and user environments is, however, disadvantageous for the client against website fingerprinting: different clients are likely to behave the same way, reducing the need of the attacker to adjust for realistic variations. We investigate this point further next.

2.1.2.1 Interaction with website fingerprinting

Tor is especially concerned with website fingerprinting. Herrmann et al. have found Tor to be the most difficult to attack, in a comparison of popular anonymity technologies [HWF09], although later authors including us have found success in website fingerprinting on Tor [CZJJ12, WG13b, WCN+14]. Tor developers have described website fingerprinting as an important open problem [Per11a, Din12].

Tor has implemented two versions of a website fingerprinting defense based on request reordering [Per11a, Per13]. We will describe this defense in Section 5.2.1.5. Furthermore, Tor's constant-size cells render some website fingerprinting techniques fruitless. As of the time of writing of this thesis, Tor developers are actively developing and seeking new defenses for Tor [Per12].

Unfortunately for Tor users, several design decisions meant to defend against other forms of privacy leakage make website fingerprinting easier than it could be.

Unified browser behaviour. To defend against browser fingerprinting attacks such as recent ones investigated by Acar et al. on Tor [AJN+13], Tor Browser attempts to behave exactly the same for all clients. It is hard to change user settings on the Tor Browser; Tor Browser resets the browser configuration file to the factory default frequently, which implies that Tor users are likely to have exactly the same settings. These factors imply that the website fingerprinting attacker does not need to adjust for different user settings that could have affected the packet sequence.

No persistent storage. Tor does not perform disk caching to protect user privacy against an attacker that may seize the client's physical assets. Since the client's cache is always empty at the start of a browsing session, she visits many pages with a cold cache. The attacker does not need to train for hot caches, which makes website fingerprinting more efficient. Furthermore, Tor does not use persistent cookies (only session cookies are allowed), so that Tor users will frequently see default index pages that are the same for all users. For example, Facebook users on Tor must log in through the front page, which has no variation and is easily identifiable.

Little noise. Tor officially discourages users from downloading videos, using torrents, and downloading large files over Tor, which are types of noise that would interfere with website fingerprinting. As a low-latency anonymity network, Tor is not suitable for downloading large files. There is active research on methods to throttle the bandwidth consumption of greedy users [ABG12, GJH13]. Tor's limited bandwidth may itself discourage users from

noise-generating behaviour. For example, if there is frequent buffering in a video, users will not want to watch it on Tor Browser.

To summarize, the Tor attacker does not need to adjust for user settings, caching, or noise. As a result, the experimenter does not need to do so either, which simplifies experimentation.

2.1.3 The Attacker

The website fingerprinting attacker is a local, passive eavesdropper who monitors the web-browsing client's network. The objective of the attacker is to identify web pages that the client is loading. In the context of web browsing, knowing the page that the client is loading is akin to knowing her web-browsing behaviour. We give a rigorous definition of the two terms:

1. A *local* attacker starts by knowing the identity of the client. The attacker does not have to be physically close to the client; for example, a proxy that the client directly connects to for web browsing is a possible local attacker for website fingerprinting; this is even if the proxy is from a different Autonomous System.

2. A *passive* attacker does not modify the traffic flow in any way. From the client's perspective, a passive attacker is impossible to detect. The client must always be prepared for the possibility of passive attackers. Since it is technically difficult to detect passive attacks in the wild, it may be more tempting to launch such an attack without fear of repercussion.

In contrast to our local attacker, *global* attackers with a large number of vantage points across the world can also apply website fingerprinting. However, we are not concerned with applying global capabilities to website fingerprinting. This is because such adversaries are already capable of attacking low-latency anonymity networks like Tor by applying timing analysis to a traffic confirmation attack [DMS04, JWJ+13].

We also do not consider *active* attackers, who may have the capability to modify or add packets in order to expose the client's destination web page. There has been interesting work on active attacks that can expose user behaviour, such as attacks that leverage active traffic-shaping strategies [GH12], remote ping detection [GKB10] and, sometimes, involve tampering with the client's device [JS12]. While interesting, active attackers rely on stronger assumptions than our passive attacker, and they are detectable.

In our work we focus especially on the ability of low-resource attackers to perform website fingerprinting on a large number of clients. We will endeavour to show that an attacker using a single desktop-class machine can de-anonymize thousands of clients simultaneously. Limiting the resources available to the attacker magnifies the threat of website fingerprinting.

2.2 Machine Learning

2.2.1 Open-World and Closed-World Analysis

Under the *open-world model*, the website fingerprinting attacker selects a set of *monitored pages* that are interesting to him. Whenever the client visits a monitored page, the attacker wants to identify the visited page. Otherwise, when the client visits a non-monitored page (any page that is not a monitored page), the attacker merely wants to identify the fact that it is non-monitored. To classify a set of testing elements, the attacker trains on a set of training elements for which he has the correct classification; this is known as supervised learning. Under the open-world model, the attacker can include some non-monitored pages in his training set, but he can never train on the non-monitored pages that the client is visiting. In other words, there is no non-monitored page which has a packet sequence in both the training set and the testing set.

When analyzing classification accuracy, we refer to a monitored page visit as a positive and a non-monitored page visit as a negative. Each specific monitored page is its own class, while the set of all non-monitored pages is a single class. If the attacker identifies the class of a testing element correctly, it is a true classification, and otherwise a false classification. We write the number of positive elements (monitored instances) as n_P and the number of negative elements (non-monitored instances) in the data set as n_N. Note that the attacker has control over n_P and n_N in the training set, and selecting them correctly is a useful strategy for website fingerprinting.

We combine "true/false" and "positive/negative" together when referring to classification accuracy. For example, a false negative event occurs when the client is visiting a monitored page and the attacker guesses that it is non-monitored. We write the number of true positives, true negatives, false positives, and false negatives as n_{TP}, n_{TN}, n_{FP}, and n_{FN} respectively. We have the following relations:

$$\begin{aligned} n_{TP} + n_{FN} &= n_P \\ n_{FP} + n_{TN} &= n_N \end{aligned}$$

Because of the above relations, we will only explicitly discuss true positives and false positives. Next, we define the true positive rate (TPR) and false positive rate (FPR):

$$\begin{aligned} a_{TPR} &= n_{TP}/n_P \\ a_{FPR} &= n_{FP}/n_N \end{aligned}$$

Let us examine a typical receiver operating characteristic (ROC) curve to illustrate the relationship between TPR and FPR. We plot such a curve in Figure 2.1, supposing we have conducted some set of experiments on some classifier to obtain the plotted results. If a classifier simply randomly guesses that the result is positive with some probability (independent of the input), and

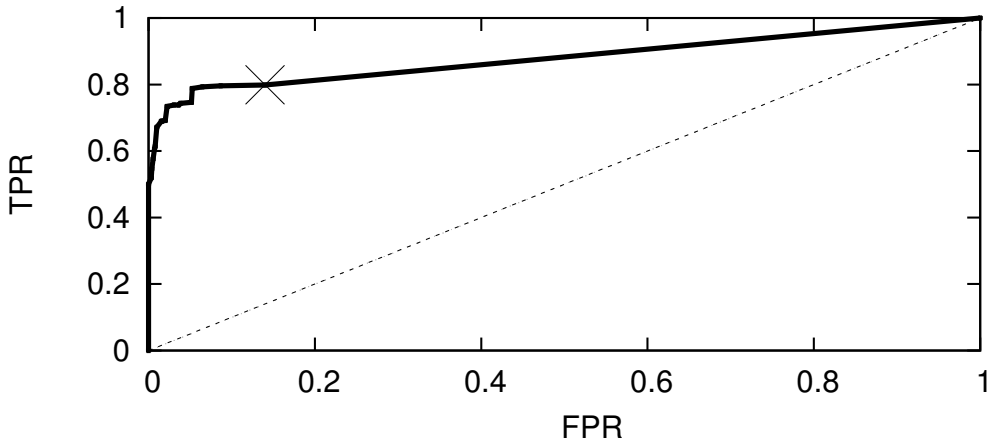

Figure 2.1: Sample ROC curve to illustrate TPR and FPR. The dotted line is the result of a classifier who simply randomly guesses that the result is positive with some probability. The crossed point is the point with the largest TPR and FPR we observed in some experiment.

when we vary this probability, the result is the dotted line. The crossed point is the point with the largest TPR and FPR we observed in our aforementioned experiments. We complete the curve by drawing a straight line from the crossed point to (1, 1). The reason we can draw such a line is because it represents a classifier's strategy of classifying using the parameters or method of the crossed point with some probability p, and simply classifying the result as positive with some probability $(1 - p)$. Similarly we draw such a line from the observed point with the lowest TPR and FPR to (0, 0).

The above is the usual formulation for binary positive/negative classification. However, our work considers a more complicated scenario, since we divide the positive class into many classes: the monitored page set consists of many different pages and the attacker must identify the correct page. If the client is visiting a monitored page, and the attacker guesses that it is a different monitored page, we call it a false positive-to-positive event (FPP). Conversely if the attacker identifies a non-monitored page as a monitored page, we call it a false negative-to-positive event (FNP). We count both of them as false positives. With this extension of the definition, we have the following relations instead:

$$\begin{aligned} n_P &= n_{TP} + n_{FN} + n_{FPP} \\ n_{FP} &= n_{FPP} + n_{FNP} \\ a_{FPR} &= n_{FP}/n_N \\ a_{FPPR} &= n_{FPP}/n_P \\ a_{FNPR} &= n_{FNP}/n_N \end{aligned}$$

While technically it is now possible for a_{FPR} to exceed 1, we will not come close to this in

Table 2.1: Example of the accuracy terminology we use in our paper. Here the attacker monitors Alexa's top 100 sites, which includes `google.com` and `facebook.com`, and does not include `uwaterloo.ca`.

True page	Attacker classification		
	`google.com`	`facebook.com`	`uwaterloo.ca`
`google.com`	True Positive	False Positive	False Negative
`facebook.com`	False Positive	True Positive	False Negative
`uwaterloo.ca`	False Positive	False Positive	True Negative

reality.

One advantage of counting both FPP and FNP as false positives is that it fits an attacker who may need to respond differently when observing different monitored pages. For example, an attacker may wish to flag users visiting some web pages for further investigation, while immediately blocking users who visit some other web pages. In that case, when the chief concern of an attacker is an inappropriate response, responding incorrectly to a non-monitored page is similar to responding incorrectly to a different monitored page: counting both as false positives accounts for such similarity. For clarity, we give an example of our use of the terms "True Positive", "False Negative", "False Positive", and "True Negative" in Table 2.1, where the attacker is monitoring all access to Alexa's top 100 sites.

In the *closed-world model*, the client never visits non-monitored pages. Correspondingly, the attacker does not need to adjust for the possibility of the client visiting non-monitored pages. This model is not realistic for website fingerprinting because there are over a billion indexed web pages, and it is not feasible for the attacker to monitor all of them. The closed-world model is purely for theoretical interest, such as when comparing classifiers. Since there are no non-monitored pages, we refer to correct classifications as true positives and incorrect classifications as false negatives, so that only the TPR is relevant.

Cai et al. describe a formula to conservatively convert the TPR of a scheme under the closed-world scenario to TPR and FPR under the open-world scenario. [CNW+14] In this work we will instead perform experiments on the open world directly to obtain accuracy values.

2.2.2 The Base Rate Fallacy

The base rate fallacy occurs when we fail to contextualize the accuracy of a classifier by disregarding the base rate; it is a failure of evaluation. As a simple example, suppose a city were

to use a seismological device that predicts whether or not a major earthquake will occur on the following day, and alerts residents in the city if so. It always detects major earthquakes, but it has a small (e.g. 1%) chance of raising a false alarm on any given day. Such a device will be overwhelmed with false alarms, because the base rate of major earthquakes in any city is low. It will fail to achieve its original goal of alerting the city's residents, who will disregard the device that cries wolf.

Because the rate at which clients access most web pages (especially a sensitive web page) is low, the base rate fallacy is relevant to website fingerprinting. Several works in the field have specifically mentioned its importance [WG13b, JAA+14, WCN+14, WG15b, WG15a]. In this section we formally analyze the base rate fallacy and explain how we adjust for it.

Suppose a website fingerprinting algorithm has accuracies a_{TPR}, a_{FPPR} and a_{FNPR} as defined above. The base rate b is the chance that the client visits a page from the monitored set; conversely $1-b$ is the chance that the client visits any other page (the non-monitored set). Then, the total fraction of pages identified as "positive" by the classifier is

$$(a_{TPR} + a_{FPPR}) \cdot b + a_{FNPR} \cdot (1 - b)$$

However, the total fraction of pages that are actually positive amongst those identified as positive is only the first term, $(a_{TPR} + a_{FPPR}) \cdot b$.

Suppose we have what appears to be a powerful classifier, with $a_{TPR} = 1$, $a_{FPPR} = 0$, and $a_{FNPR} = 0.01$, which monitors a single sensitive web page. Now let $b = 0.001$ for this web page. Then amongst all pages identified as positive by the classifier, the proportion of positive classifications that actually originated from this sensitive page is $(1+0) \cdot 0.001 / ((1+0) \cdot 0.001 + 0.01 \cdot 0.999) \approx 1/10$, about one-tenth. In this example, if the attacker were to block all users trying to visit this web page, it would block nine non-sensitive pages for each correctly blocked sensitive page. Such a significant rate of collateral damage may not be acceptable to the attacker.

Assuming a classifier with relatively high TPR and low FPR, as is true for our latest website fingerprinting attacks, the effect of the base rate fallacy can be approximated by the ratio between a_{FNPR} and b. If a_{FNPR} is an order of magnitude or more lower than b, then the base rate fallacy does not apply. In the above example, a_{FNPR} is equal to b, so about one-half of identified positive accesses were not positive. If it were much higher than b, then the base rate fallacy would overwhelm the classifier. Therefore when judging the classifiers described in this work, we emphasize that we cannot possibly claim success with all b. The lower b is, the harder the classification problem.

Note that only b is relevant in the above analysis; holding b constant, the number of pages in the monitored or non-monitored set is not relevant. The fact that there are over a billion indexed web pages does not mean we need to fingerprint a significant portion of them.

2.2.3 Training and Testing

To train and test our classifiers, we divide the collected data set into 10 equal parts, and we set aside one part for testing and the rest for training. This process is repeated 10 times, cycling the testing part through each of the 10 parts one at a time. This testing method is known as 10-fold cross-validation, and we use this method to compare website fingerprinting attacks with each other in Section 3.4.1. To ensure the correctness of the open-world model as described in Section 2.2.1, in which we required that the training and testing non-monitored pages are never the same, we collect only one packet sequence for each non-monitored page.

When the testing part includes only one packet sequence, rather than one of the 10 parts of the data set, the testing method is referred to as leave-one-out cross-validation Leave-one-out classification requires more computation than 10-fold cross validation as the classifier must be trained once for each element in the data set rather than a total of 10 times. We use leave-one-out cross-validation specifically for `Wa-kNN` as its k-Nearest Neighbours classifier requires no training. This increases the accuracy of `Wa-kNN` and gives us a better understanding of the attacker's potential accuracy. Still, for comparison with other classifiers in Section 3.4.1, where not all classifiers can practically be tested with leave-one-out cross-validation, we test all classifiers with 10-fold cross validation.

2.3 Terminology

2.3.1 General Terminology

We write a list M as $M = \langle M_1, M_2, \ldots, M_{|M|} \rangle$; the i-th element of M is M_i, and the length of the list M is $|M|$. For a list, the order matters, whereas the order does not matter for sets $S = \{S_1, S_2\}$. We write $M_{[:i]}$ as the first i elements of M, and we say that $M_{[:i]}$ is a prefix of M.

Given a function f which transforms a list M into a new list, we write the output as $f(M) = \langle f(M)_1, f(M)_2, \ldots, f(M)_{|M|} \rangle$.

We reserve m and M generally to temporarily denote terms that will not be used again in other contexts, so their meaning does not carry over between sections.

2.3.2 Website Fingerprinting Terminology

A packet sequence is a list, which we write as $P = \langle (t_1, \ell_1), (t_2, \ell_2), \ldots, (t_{|P|}, \ell_{|P|}) \rangle$. Each t_i is the *interpacket time* between the $(i-1)$-th packet and the i-th packet, with $t_1 = 0$. Each ℓ_i is

the *length* and *direction* of the packet; the sign indicates direction, positive for outgoing packets from the client and negative for incoming packets, and $|\ell_i|$ is the length. To refer to the i-th packet of P, we write P_i; in other words we say that the position of packet P_i in P is i. To refer to the i-th packet length, we can write P_{ℓ_i} or simply ℓ_i.

Sometimes, rather than the interpacket time, we would like to refer to the total elapsed time of some packet P_i (amount of time between P_1 and P_i), in which case we use the capital letter T_i; for example, $T_{|P|}$ is the total duration of the packet sequence.

We note especially that the terms *sequence length* and *packet length* should not be confused. The sequence length is $|P|$, the number of packets in the packet sequence. The packet lengths are ℓ_i, the number of bytes in each packet. We will discuss sequence padding and packet padding in Chapter 5. Sequence padding means adding packets to the sequence, and packet padding means adding bytes to each packet.

In general, to refer to a list derived from P, we write its type as a subscript to P. We use the following notation:

1. P_t is the list of interpacket times of P in the same order, e.g. $P_{t_1} = 0$.
2. P_ℓ is the list of packet lengths of P in the same order.
3. P_U is the list of unique packet lengths of P. No element in P_U appears more than once. The list is sorted in ascending order of packet length; this includes negative packet lengths (i.e. incoming packets to the client), so P_{U1} is the size of the largest incoming packet.
4. P_+, P_- are the outgoing and incoming packets of P respectively. We can combine them with U, for example, so that P_{U+} is a list of unique outgoing packet lengths of P, in ascending order of packet length.

We note that it is possible for a packet sequence to be identical to another (a *collision*). While this is extremely unlikely normally, we will design defenses which cause collisions in Chapter 5.

On Tor, we use cell sequences rather than packet sequences. We described in Section 2.1.2 how Tor packages data in cells. The attacker is able to parse packet sequences into Tor cell sequences by reading TCP/IP headers (which are not encrypted). In cell sequences, we write both incoming and outgoing cells as length one; for example, $P_U = \langle -1, 1 \rangle$ for a typical P written as a cell sequence. For simplicity in this work, we overload the phrase "packet sequence" to also refer to cell sequences when we are in the context of Tor.

We write the full data set of all packet sequences as S. Sometimes, we specify the training set as S_{train} and the testing set as S_{test}. To distinguish between the closed-world and open-world

scenario, the number of monitored pages is n_{mpage}, the number of training instances for each monitored page is n_{minst}, and the number of non-monitored pages in the training set is n_{nmpage} (for which there is always 1 instance each). We use k-fold testing to compute classification accuracy, so all elements in S will be tested; the size of the testing set is the same as that of the training set.

The class of a packet sequence P is $c(P) \in \{0, 1, \ldots, n_{mpage}\}$. Each packet sequence belongs to one and exactly one class. We write the set of all packet sequences belonging to class i as $C_i = \{c(P) = i : P \in S\}$. C_0, class 0, is the non-monitored class of all web pages that the attacker does not monitor.

We can therefore phrase the attacker's objective as

Given $(P_{train}, c(P_{train}))$ for each $P_{train} \in S_{train}$, predict $c(P_{test})$ for each $P_{test} \in S_{test}$.

2.4 Data Collection

As an application of machine learning, website fingerprinting is fundamentally rooted in data collection. Whether or not our data represents the real situation is a significant determinant of the quality of our work. In this section, we describe the steps we take to collect our data and ensure that our data set would give meaningful results.

2.4.1 Data Collection with a Browser

We used Firefox 38.0 to collect data. To start loading `www.example.com`, we call Firefox with the command:

```
> firefox www.example.com
```

We collect the corresponding packet sequence by using `tcpdump` to log all packets to and from the HTTP and HTTPS ports during page load. We load each page for 15 seconds. After this amount of time, we kill the browser process and start a new process to load the next page. We kill the process because the browser sometimes has warnings and dialogs that may halt data collection.

We do not keep a cache; our data set corresponds to a client whose browser cache is always cold. Since many privacy-enhancing technologies such as Tor Browser and private browsing do

not cache to permanent storage, we can reasonably assume that the client will frequently visit pages without the benefit of caching.

Before page loading, we also keep a backup of the configuration file of the browser, and we replace the configuration file with the backup every time between page loads. This is to ensure that our configuration file is the same for all packet sequences. For example, Tor Firefox frequently resets the configuration file as a defense against possible malicious software that edits the configuration file to disrupt the client's privacy, but also unfortunately disrupts our data collection process.

2.4.2 Data Collection on Tor

On Tor Browser, we collect cell sequences instead of packet sequences. We let the browser run for 60 seconds; this is because we found that the mean page load time on Tor is around 15 seconds, which we also reported in previous work [WCN+14]. Our chosen time should be far larger than the mean page load time on Tor in order to capture almost all cells.

To perform data collection on Tor Browser, we must take into consideration the specific design of Tor (see Section 2.1.2).

2.4.2.1 Circuit construction

Tor relays have a wide range of bandwidths. As of September 2015, the top 1% of relays comprised 19% of the total bandwidth of Tor and the top 20% of relays comprised 86% of the bandwidth. Bandwidth and congestion will affect packet sequences, so that traffic instances collected using the same circuit are more similar than those collected using different circuits. We assume that the attacker can observe the client's network traffic, but cannot control or observe which Tor circuits the client is using (this is a basic assumption necessary for the privacy guarantees of Tor [DMS04]). We should not use the same circuit for both training and testing, as that would give an unrealistic advantage to the attacker.

By default, Tor can only use a circuit for up to ten minutes, after which Tor will not launch new connections on the circuit. Suppose we want to collect n_{minst} instances of each web page in our data set. Two ways to do so are as follows: we can visit each page once, and repeat this n_{minst} times; or we can visit each page n_{minst} times, and then do so for each page. If we were to collect data using the latter method, then instances of the same site are more likely to be accessed with the same circuit, while instances of different sites are more likely to be accessed with different circuits. The inherent difference of circuits imposes a difference upon the traffic

instances of different sites, which could help the machine learning algorithm separate them, giving the attacker an unrealistic advantage. Therefore, the correct way to visit web pages is the former method, which we adhere to.

Although we can identify circuit construction cells from each cell sequence, we do not remove them. Circuit construction cells are part of the noise that realistic clients would generate during web browsing; the attacker will simply have to adjust for them.

2.4.2.2 Network conditions

The same page can appear as different packet sequences under different network conditions. A higher bandwidth, for example, will reduce the total amount of time needed to load the packet sequence. As argued in Section 2.1.2.1, we are comfortable with using one client located at a fixed IP to collect our Tor data set. Tor circuits last for 10 minutes each, while our data set took several days to collect; the difference implies that our data set has a broad view of the network.

Tor uses entry guards for the first node of a circuit: the same entry guard lasts for 30 to 60 days for the same client. If we were to randomly choose a slow entry guard that acts as a bottleneck, all of our packet sequences will be affected. Thus, we also disable entry guards for data collection so that our data will be more representative of the entire network. Disabling entry guards simply means that the entry node will be randomly chosen from the set of possible guards every time we build a new circuit.

2.4.2.3 Localization

Depending on the location of the exit relay, a web page could present entirely different data. This is known as website localization and it is especially prevalent among sites with international reach, such as `google.com` and `yahoo.com`. Roughly speaking, there are two types of website localization that are of concern to us. The first type is redirection: a Canadian client attempting to access `google.com` will be redirected to `google.ca` by Google—the site that the client is actually accessing is different depending on locality. The second type involves content changes: a German advertisement may be shown to a person connecting from Germany or a Tor client exiting from Germany, but it is less likely for other clients to see the German advertisement. With Tor, the location of the exit relay determines the apparent location of the client. To protect the client's privacy, Tor does not attempt to choose the exit relay based on the client's location.

In a previous work [WG13b], we attempted to adjust for localization effects by specifying which web page localization version to visit in the data set (for example, we would visit

`google.ca` specifically). Our rationale was that we wanted to ensure the web page stayed the same. Juarez et al. criticized this methodology as it gave the attacker an unrealistic advantage [JAA+14]. The attacker should not be able to predict the circuit of the client and therefore her localization version. We agree with Juarez et al. and we do not attempt to adjust for localization in our data set in this thesis.

2.4.3 Data Sets

Applying the methodology in Section 2.4, we collected the following major data sets used across several sections:

1. Non-Tor data set: We used Firefox 38.0 to visit web pages. Most privacy technologies (such as VPNs and proxies) do not attempt to defend the packet sequence, so results on this data set would be similar to results on most privacy technologies. We collected 100 instances of Alexa's top 100 pages as the set of monitored pages, and 9000 instances of Alexa's top 10000 pages (1 instance of each page, and discarding failed pages) as the set of non-monitored pages. We collected this data in July 2015.

2. Tor data set: We collected our data on Tor Browser 4.5a4 with Tor 0.2.7.0-alpha. As above, we collected the same number of monitored and non-monitored pages. Tor is especially interesting to us because it has applied defenses against website fingerprinting and continues to improve its defenses (see Section 5.2.1.5). We collected the data in June 2015.

In our previous work [WCN+14] we collected a data set of sensitive pages banned by several ISPs in various countries. We do not do so in this thesis because we have found that these sites are taken down frequently, which makes it harder to reproduce our results. Thus, we will only use Alexa's top sites. We also collected several other data sets for use in specific sections, and we will describe these data sets in detail in the relevant sections.

Chapter 3

Attacks

> "Rapidity is the essence of war: take advantage of the enemy's unreadiness, make your way by unexpected routes, and attack unguarded spots."
>
> *The Art of War*

Eavesdropping on a web-browsing client, the attacker has collected packet sequences corresponding to web page visits, and he seeks to determine which web pages the client is visiting. To classify packet sequences to web pages, he utilizes machine learning techniques. We examine website fingerprinting attacks in this chapter.

In general, each classifier is characterized by its answer to the following question:

How are packet sequences from the same page similar to each other?

The success of an attack depends on its ability to identify differences between packet sequences that indicate that they came from different web pages, while at the same time tolerating differences between packet sequences that come from the same page. We describe previous website fingerprinting attacks, as well as our own new attacks, in Section 3.1.

Some attacks use distance metrics to measure packet sequence similarity. Distances can be computed either on feature sets of packet sequences, or the sequences themselves. More similar packet sequences are more likely to belong to the same class. We describe distance metrics in Section 3.2. In particular, we describe our new distance metric learning algorithm, Weight Learning by Locally Collapsing Classes (WLLCC), in Section 3.2.5. WLLCC calculates an effective distance metric for the k-Nearest Neighbours (k-NN) classifier.

Each attack implicitly or explicitly uses a set of features from packet sequences to classify elements. In Section 3.3, we categorize these features into four groups: unique packet lengths, sequence lengths, packet ordering, and interpacket timing. We analyze each attack based on this categorization to show a general trend of shifting from interpacket timing, to unique packet lengths, and finally to sequence lengths and packet ordering.

We evaluate and compare website fingerprinting attacks in Section 3.4. We do so on both the Tor and non-Tor data sets. We will find that our newest attacks can achieve high classification accuracy with a very low false positive rate on both data sets. Although Tor attempts to protect itself against website fingerprinting, we will show that its protection is weak. Computationally, we are capable of attacking thousands of web-browsing clients simultaneously from a single desktop-class computer. In doing so, we demonstrate that website fingerprinting attacks are accurate under laboratory settings.

In the wild, however, conditions may vary and significant sources of noise may deteriorate classification accuracy. In the next chapter, Chapter 4, we examine the difference between realistic conditions and laboratory conditions, and we describe several techniques to allow website fingerprinting attacks to operate realistically. We will see that website fingerprinting is indeed a realistic threat to user privacy, which motivates the succeeding chapter, Chapter 5, on website fingerprinting defenses.

3.1 Description of Website Fingerprinting Attacks

Researchers have published WF attacks since 1998, initially under the wider umbrella term "traffic analysis". Since then, new WF attacks have succeeded in classifying web-browsing traffic under increasingly weaker assumptions, thus highlighting the threat of WF to web-browsing privacy.

In this section, we list known WF attacks and describe the classification process in each classifier. We give each attack a name written in `typewriter font` so as to distinguish them from other attacks and from the names of their underlying classifiers. The first two letters of each name are the first two letters of the name of the first author in the published work. Amongst these attacks, we designed the three newest attacks in this list: `Wa-OSAD`, `Wa-FLev`, and `Wa-kNN`. We implement twelve website fingerprinting attacks (all attacks in this section except the resource length attacks in Section 3.1.1), and compare them in Section 3.4. The code for all twelve attacks is available for download; see Appendix A.

3.1.1 Resource Length Attacks

Some of the earliest WF attacks assumed that web resource lengths were revealed to the attacker, such as Cheng et al. in 1998 [CA98], Sun et al. in 2002 [SSW+02], and Hintz in 2003 [Hin03]. These works start with the assumption that the attacker knows the resource length of each resource in a web page. This is because, without the persistent connections offered by HTTP/1.1, older technologies such as SafeWeb [Hin03] expose individual resource lengths to an attacker who can distinguish between TCP connections, as the data for one connection corresponds to one resource.

Most modern privacy-enhancing technologies, such as SSH tunneling and Tor, do not leak resource lengths. We have not been able to extract resource lengths from packet sequences on newer technologies. All newer attacks do not assume that the attacker is able to determine resource lengths. Therefore, we will not assume knowledge of resource lengths either.

Nevertheless, the observation that unique resource lengths are important is deeply connected to the notion of unique packet lengths, which is useful for many of the following WF attacks.

3.1.2 Cross-correlation with Timing Windows (Sh-Timing)

Shmatikov and Wang[1] (2006) described a new defense for website fingerprinting [SW06]. They noted that previously proposed defenses are either ineffective or inefficient. We will discuss their defense in Section 5.2.1.1.

To show the effectiveness of their defense, they developed a new website fingerprinting attack against it, `Sh-Timing`. This passive attack was adapted from a work (not itself about website fingerprinting) previously published by Levine et al. in 2004 [LRWW04]. This makes `Sh-Timing` one of the earliest website fingerprinting attacks that worked without presuming knowledge of resource lengths.

The cross-correlation between two ordered lists $P = \langle p_1, p_2, \ldots \rangle$ and $Q = \langle q_1, q_2, \ldots \rangle$ is:

$$X(P,Q) = \frac{\sum_{i=1}^{\min\{|P|,|Q|\}} (P_i - \overline{P})(Q_i - \overline{Q})}{\sigma_P \sigma_Q}$$

where $\overline{P}$, $\overline{Q}$, σ_P and σ_Q are the means and standard deviations of P and Q respectively. A higher cross-correlation score indicates greater similarity.

[1] Ming-Hsiu Wang.

For each packet sequence P, the algorithm derives P_w: P_w is a series of packet counts P_{w_i}, where each P_{w_i} is the number of packets (totalling both directions) between the $(i-1)$-th second and the i-th second.

To classify packet sequence P, `Sh-Timing` computes $X(P_{windows}, P'_{windows})$ for each training set packet sequence P'. `Sh-Timing` classifies P to the class of the highest-scoring training element. Shmatikov and Wang suggest that a threshold scheme might reduce false positives, but as they did not present a method to determine the threshold, we do not implement `Sh-Timing` with a threshold.

3.1.3 Cross-correlation with Interpacket Times (Bi-Intertiming)

Bissias et al. (2006) published a website fingerprinting attack that heavily employs interpacket timing [BLJL06], `Bi-Intertiming`. This attack is quite similar to `Sh-Timing`, but instead of packet counting, it uses interpacket times. We continue the terminology for `Sh-Timing` to describe `Bi-Intertiming`.

Recall from Section 2.3 that $P_t = \langle t_1, t_2, \ldots, t_{|P|} \rangle$ is the sequence of interpacket times for each packet sequence P. We extend this definition to classes, so that $C_t = \langle t_1^C, t_2^C, \ldots \rangle$, where t_i^C is the mean interpacket time of the i-th packet in each packet sequence in C. We can do the same for P_ℓ (the sequence of packet lengths) to obtain C_ℓ.

To classify packet sequence P, `Bi-Intertiming` computes $X(P_t, C_t)$ and $X(P_\ell, C_\ell)$ for each class C. Bissias et al. recommend multiplying the two cross-correlation scores for interpacket times and packet lengths.

3.1.4 Jaccard Coefficient (Li-Jac)

Liberatore and Levine (2006) published two of the first website fingerprinting attacks that were highly accurate without knowing resource lengths [LL06]. These attacks demonstrated that unique packet lengths help classify a web page. Packets are almost always sent at the Maximum Transmission Unit (MTU) whenever possible, which is generally 1500 bytes on Ethernet, and therefore packet lengths which are not the MTU indicate the remainders of resource lengths (modulo the MTU).

The first attack they proposed, `Li-Jac`, relies on the Jaccard coefficient. We denote P_U as the list of all unique packet lengths for some sequence P. We then extend this definition to classes, so that each class C has a set of unique packet lengths C_U, where $\ell \in C_U$ if ℓ is present in the *majority* of packet sequences $P \in C$. As usual with packet lengths, the direction is included

as a positive/negative sign, positive indicating outgoing from the client and negative indicating incoming.

The Jaccard coefficient is a measurement of the similarity of two different sets P^1 and P^2 as follows:

$$J(P^1, P^2) = \frac{|P^1 \cap P^2|}{|P^1 \cup P^2|}$$

To classify P, we measure $J(C_U, P_U)$ for all classes C, and classify P as the highest-scoring class. Two packet sequences are similar if they share a large number of unique packet lengths.

3.1.5 Naïve Bayes (Li-NBayes)

The second attack Liberatore and Levine proposed was a Naïve Bayes classifier, `Li-NBayes`, based on packet lengths and the number of times they appeared within each packet sequence.

Suppose we want to decide if packet sequence P belongs to class C. From P, we extract $P_S = \{(\ell_1, f_{\ell_1}), (\ell_2, f_{\ell_2}), \ldots, (\ell_n, f_{\ell_n})\}$. Here, each $\ell_i = P_{U_i}$ is a unique packet length and f_{ℓ_i} is the number of times ℓ_i appears in P. We extend the definition of P_S to classes, so that for class C:

$$C_S = \{(\ell_1, F_{\ell_1}), (\ell_2, F_{\ell_2}), \ldots\}$$

F_{ℓ_i} are multisets of all f_{ℓ_i} of each $P \in C$. If there are $|C|$ training packet sequences in class C, then each F_{ℓ_i} will have length $|C|$.

`Li-NBayes` applies the Naïve Bayes assumption as follows: it assumes that the packet lengths ℓ_i and their frequencies f_{ℓ_i} occur independently of other packet lengths and frequencies, so the probability and the score assigned to $P \in C$ is:

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

`Li-NBayes` computes the probability $p(f_{\ell_i} \in F_{\ell_i})$ with kernel density estimators (KDEs). A KDE is essentially a histogram with bins (kernels) that stack on top of each other. We can compute a KDE from F_{ℓ_i} for each ℓ_i; however, the length of each bin (called kernel bandwidth for KDEs) can significantly affect performance. `Li-NBayes` uses a kernel bandwidth selection metric called AMISE. We refer the reader to Scott [Sco15] for a more detailed writeup of KDEs and the intricacies of kernel bandwidth selection.

3.1.6 Multinomial Naïve Bayes (He-MNBayes)

Herrmann et al. (2009) proposed a number of improvements to `Li-NBayes` by incorporating techniques from text mining known to improve the accuracy of matching documents (web pages) when searching with keywords (unique packet lengths) [HWF09]. They use the Multinomial Naïve Bayes classifier, and transformed the input using the term frequency (TF), inverse document frequency (IDF), and cosine normalization (CN) schemes.

`He-MNBayes` also uses the Naïve Bayes assumption, with a different way to compute $p(f_{\ell_i} \in F_{\ell_i})$. Instead of using the KDE, they compute $p(f_{\ell_i} \in F_{\ell_i})$ as follows:

$$p(f_{\ell_i} \in F_{\ell_i}) = \left(\frac{F_{\ell_i}}{\sum F_{\ell_i}} \right)^{f_{\ell_i}}$$

We can interpret the above as taking the Naïve Bayes assumption one step further: the occurrence probability of each individual packet is independent of all other packets, including packets of the same length.

Herrmann et al. applied TF, IDF, and CN in order. TF, IDF and CN modify f_{ℓ_i}, the exponent term of the above formula. This modification only applies to the above probability computation; it does not change f_i for the purposes of building F_ℓ for each ℓ.

Term Frequency (TF). TF transforms the power logarithmically:

$$TF(f_{\ell_i}) = \log(f_{\ell_i} + 1)$$

Inverse Document Frequency (IDF). IDF discards common packet lengths, magnifying the effect of unique packet lengths. Suppose $S_{\ell_i} \subseteq S_{train}$ is the set of training packet sequences that contain the packet length ℓ_i. Then:

$$IDF(f_{\ell_i}) = f_{\ell_i} \cdot \log \frac{|S_{train}|}{|S_{\ell_i}|}$$

For example, if all packet sequences contain incoming MTU packets, then $|S_{\ell_i}| = |S_{train}|$, so $IDF(f_{\ell_i}) = 0$ for all packet sequences; the number of times f_{ℓ_i} occurred would not matter. For this reason, `He-MNBayes` does not work well on Tor, which uses fixed-size cells.

Cosine Normalization (CN). CN modifies the frequency by dividing the frequency with the Euclidean norm of all frequencies:

$$CN(f_{\ell_i}) = \frac{f_{\ell_i}}{\sqrt{\sum_j f_{\ell_j}^2}}$$

Herrmann et al. argue that one advantage of `He-MNBayes` over `Li-NBayes` was that, because of CN, `He-MNBayes` uses relative frequencies of packet lengths rather than the absolute frequency.

Herrmann et al. showed that `He-MNBayes` achieved a higher accuracy than both `Li-Jac` and `Li-NBayes` in comparable experiments. However, they also demonstrated that their attack is ineffective on Tor; this is because all Tor cells have a fixed size of 512 bytes (as described in Section 2.1.2).

3.1.7 Levenshtein Distance on Unique Packet Lengths (Lu-Lev)

Wright et al. proposed a WF defense we refer to as Morphing (see Section 5.2). In response, Lu et al. (2010) published an attack that is capable of distinguishing morphed packet sequences [LCC10], `Lu-Lev`. It does so by heavily focusing on packet ordering. `Lu-Lev` relies on the observation that the order of non-MTU packets does not often change between packet sequences derived from the same web page.

From each packet sequence P, `Lu-Lev` extracts $P_{\ell+}$ and $P_{\ell-}$, the list of outgoing and incoming packet lengths respectively, where the order is the same as in P. `Lu-Lev` ignores MTU packets (size 1500) when constructing $P_{\ell+}$ and $P_{\ell-}$.

To classify a testing packet sequence P, `Lu-Lev` compares the similarity between the testing packet sequence and each training packet sequence P', and assigns the testing packet sequence to the class of the most similar training packet sequence. In other words, only the most similar training packet sequence determines classification. `Lu-Lev` defines the similarity between P and P' as:

$$1 - 0.6 \cdot d_{lev}(P_{\ell+}, P'_{\ell+}) - 0.4 \cdot d_{lev}(P_{\ell-}, P'_{\ell-})$$

d_{lev} is the Levenshtein distance, which is equal to the number of insertions, deletions and substitutions of packet lengths to transform one input packet sequence into another (more in Section 3.2.1). d_{lev} is further normalized by the length of the longer input sequence.

`Lu-Lev` computes the Levenshtein distances between outgoing and incoming packets separately, and then combines them linearly with weights 0.6 and 0.4. Lu et al. chose these weights as they produced good results, and demonstrated that their attack is able to achieve WF at a high accuracy and defeat Morphing. The authors also pointed out the importance of the open-world scenario (which they called the problem of detection). Although `Lu-Lev` depends on packet order, it does not examine the ordering between outgoing and incoming packets ($P_{\ell+}$ and $P_{\ell-}$ are separate).

3.1.8 Feature Extraction on SVM (Pa-FeaturesSVM)

Panchenko et al. (2011) published an attack, `Pa-FeaturesSVM`, that specifically targeted web browsing clients that use anonymity technologies including JAP (Java Anonymous Proxy) and Tor, as Herrmann et al. were not able to successfully perform WF on these technologies [PNZE11]. As unique packet lengths were a known feature for website fingerprinting, JAP and Tor intentionally obscure unique packet lengths by padding traffic to fixed-size units. JAP uses 998-byte chunks and Tor uses 512-byte cells.[2]

Panchenko et al. therefore leveraged a number of other features, based on expert knowledge of how web pages function. For example, one feature, the "HTML document size", attempts to estimate the size of the main HTML document by taking the number of incoming packets between the first outgoing GET request and further outgoing packets. They also append all packet sequence lengths P_ℓ to the feature list. Panchenko et al. describe their feature list in detail [PNZE11]; we refer the reader to their work for a full description.

`Pa-FeaturesSVM` uses a Support Vector Machine (SVM). In its most basic form, SVM attempts to find a best-fitting linear separator between two classes. To do so, it computes distances between points from each class, based on sets of features extracted from those points. The closest points to the linear separator are the "support vectors"; removing all points except the support vectors will not change SVM.

To expand the power of SVM, we can compute distances with kernel methods, so that points appear as if they are in higher dimensions; the linear separator thus will also elevate in dimension. `Pa-FeaturesSVM` uses a radial basis function (RBF) kernel to compute distances (we describe this in greater detail in Section 3.2.3). A cost variable can also tolerate error in SVMs, allowing SVMs to perform well with noisy data. While SVM is expensive in its basic form because it requires the inverse of a large matrix of distances between all pairs of points, the matrix itself is often sparse, and practical implementations (such as LIBSVM [CL11]) use powerful

[2] Panchenko et al. perform experiments on both JAP and Tor, but we will only show results on Tor.

methods to optimize sparse matrix inversion. (More details on SVMs are given by Vapnik and Chervonenkis [VC74].)

Panchenko et al. demonstrated that `Pa-FeaturesSVM` is powerful enough to achieve WF on Tor and JAP at an accuracy comparable to attacks on clients who are not using these technologies, and they were the first to do so. They used up to 800 pages for training and testing. They also showed that their attack performed well in a small open-world scenario with two classes C_0 and C_1, where the attacker is monitoring a single web page C_1 with a binary classifier.

3.1.9 Variable N-Gram (Dy-VNG)

Dyer et al. (2012) described a variable n-gram classifier, `Dy-VNG`, in their work [DCRS12]. `Dy-VNG` applies the Naïve Bayes classifier to three features: total time, total length of all packets in each direction, and burst lengths.

Burst lengths are defined as follows. From P_ℓ, the sequence of packets lengths of P, we merge together all adjacent packets in the same direction by adding their lengths together. The results are burst lengths in alternating directions. Dyer et al. observe that burst lengths are similar to n-grams used in natural language processing.[3]

It is interesting to note that on Tor specifically, burst lengths are inaccurate. This is because Tor uses SENDME cells for flow control, as mentioned in Section 2.1.2. SENDMEs interrupt bursts, as they flow in the opposite direction of the current burst. It is also non-trivial to identify and remove SENDMEs; we attempt to do so in Section 4.5.2. Current works on website fingerprinting leave SENDMEs in the trace. This problem affects other website fingerprinting attacks that use burst lengths as well, such as `Wa-kNN` (described later).

As it was not specified in their work, we implemented `Dy-VNG` with KDEs, much like `Li-NBayes`. We trained a KDE for each of the three features: total time, total length of all packets in each direction, and burst lengths. When classifying a testing element, we extracted those three features and obtained a probability estimate from each respective KDE. Finally, we multiplied the three results to obtain the Naïve Bayes probability estimate.

Dyer et al. compared `Dy-VNG` with `Pa-FeaturesSVM` under a number of scenarios and did not find their attack to be an improvement in any scenario. They suggest that this is because `Pa-FeaturesSVM` uses a more complicated classifier, SVM, with more fine-grained features. We point out that the use of total time in `Dy-VNG` may lead to inaccuracy considering that the attacker cannot predict the client's network conditions.

[3] Burst lengths ignore adjacent order, unlike n-grams. For example, "ab" and "ba" are different bi-grams, but transposing two adjacent packets in the same direction would not change the burst length.

3.1.10 OSAD for SVM Kernel (Ca-OSAD)

Cai et al. (2012) improved the accuracy of WF on Tor by adding transpositions to the list of Levenshtein operations [CZJJ12]. Their distance is known as the Optimal String Alignment Distance (OSAD), so we call their attack `Ca-OSAD`. We give more details on OSAD and other edit distances in Section 3.2.1; for now it suffices to say that $d_{OSAD}(P, P')$ computes a distance between packet sequences P and P', where a lower score indicates greater similarity. Cai et al. normalize d_{OSAD} such that it ranges from 0 to 1.

To classify a testing packet sequence P, `Ca-OSAD` computes $d_{OSAD}(P, P')$ with each training packet sequence P'. This distance d_{OSAD} replaces the kernel distance normally used in SVMs, such as the RBF distance used in `Pa-FeaturesSVM`. Since it eschews the kernel method, `Ca-OSAD` does not have the ability to project data points onto higher dimensions, but nevertheless it achieves a high WF accuracy using d_{OSAD}.

Cai et al. round each TCP packet length upwards to the nearest multiple of 600 in order to estimate the number of Tor cells transmitted. This is similar (but less accurate) than our use of Tor cell sequences by directly parsing packet sequences. They were successful in attacking and defeating several published defenses, including Tor's Pipeline defense (described in Section 5.2.1.5).

3.1.11 Modified OSAD for SVM Kernel (Wa-OSAD)

We further improved the accuracy of `Ca-OSAD` on Tor by incorporating a number of modifications into OSAD [WG13b] (2013). We call our attack `Wa-OSAD`. Instead of TCP packet sequences, we extracted Tor cell sequences directly from the packet capture by reconstructing TCP streams and parsing the SSL records contained therein. We also attempted to identify and remove SENDME cells. We describe the SENDME removal method with experimental results in Section 4.5.2.

We made other modifications to the distance computation, $d_{OSAD}(P, P')$ as follows:

1. We removed substitutions from the list of Levenshtein operations as these do not correspond to the variations caused by loading the same page repeatedly.

2. We increased the cost for outgoing packet operations as it is less likely for the same page to have a different number of outgoing packets than a different number of incoming packets. The number of outgoing packets is closely related to the total number of resources.

3. We varied the transposition cost across the packet sequence, so that the transposition cost is larger near the beginning of the packet sequence and it is lower at the end. This is because packets at the start of the sequence are less prone to random changes due to network conditions.

We explain the intuition behind these modifications and evaluate their effect on WF classification accuracy in Section 3.4.3. Similar to `Ca-OSAD`, we use the modified distance computed above to replace the kernel of an SVM.

3.1.12 Fast Distance for SVM Kernel (Wa-FLev)

In the same work, we showed another attack, `Wa-FLev`, with a lower accuracy than both `Ca-OSAD` and `Wa-OSAD` under comparable experiments, but which was around three orders of magnitude faster to train and test. Rather than finding the least-cost sequence of insertions, deletions, and transpositions, `Wa-FLev` finds the least-cost sequence for insertions/deletions and transpositions separately, and then add the two together. We describe this process as follows.

The calculation of d_{flev} takes two parameters $cost_{trans}$ (the cost of transpositions) and $cost_{del}$ (the cost of deletions or insertions) as input. Given two packet length sequences P^1_ℓ and P^2_ℓ, we compute $d_{flev}(P^1_\ell, P^2_\ell)$. First, we mark all elements in P^2_ℓ as "unused" and set d_{flev} to 0.

Transpositions: We enumerate across all packet lengths $\ell^1_i \in P^1$. For each ℓ^1_i, we find the first identical element $\ell^1_i \in P^2_\ell$ that is unused; suppose $\ell^2_j = \ell^1_i$. We increment d_{flev} by $|j - i| \cdot cost_{trans}$. Then, we mark ℓ^2_j as used. We move on to the next packet length in P if we cannot find an unused $\ell^1_i \in P^2_\ell$.

Insertions/Deletions: After the above, we are left with a number of unused packets in P^1_ℓ and P^2_ℓ. We take the total number of packets left in both packet sequences, multiply the number by $cost_{del}$, and add it to d_{flev}. This represents the number of insertion/deletions necessary to merge the two sequences. Let us denote $P_{\ell+}$ as the number of outgoing packets in P, and similarly $P_{\ell-}$ as the number of incoming packets. In the case where all packets have the same length, as in Tor, the number of insertion/deletions necessary is equal to $|P^1_{\ell+} - P^2_{\ell+}| + |P^1_{\ell-} - P^2_{\ell-}|$.

We can also build a dictionary to speed up the above attack. For each packet length ℓ in P^1_U, the list of unique packet lengths of P^1, we build a dictionary $Dict^\ell$. Let the i-th element of $Dict^\ell$ be $Dict^\ell_i$, as usual with list notation. Every $Dict^\ell_i \in Dict^\ell$ satisfies $P^1_{Dict^\ell_i} = \ell$. $Dict^\ell$ is empty

Algorithm 1 Fast Levenshtein-like distance

Input: Strings P^1, P^2, and dictionaries $Dict^\ell$ for locations of each packet length $\ell \in P^1_U$; insertion/deletion cost $cost_{del}$, transposition cost $cost_{trans}$

Output: $d_{flev}(P^1, P^2)$

Initialize $d_{flev} = 0$
for $0 < i \leq |P^2_\ell|$ **do**
  if $Dict^{P^2_{\ell_i}} \neq \emptyset$ **then**
    $d_{flev} = d_{flev} + |Dict_1^{P^2_{\ell_i}} - i| \cdot cost_{trans}$
    $Dict^{P^2_{\ell_i}} = Dict^{P^2_{\ell_i}} \backslash \{Dict_1^{P^2_{\ell_i}}\}$
  else
    $d_{flev} = d_{flev} + cost_{del}$
  end if
end for
$d_{flev} = d_{flev} + \sum_\ell |Dict^\ell| \cdot cost_{del}$
Return d_{flev}

if $\ell \notin P^1_\ell$. The dictionary is sorted in ascending order. This is a simple algorithmic optimization that saves us time in the "Transpositions" step above. Computation of the dictionary is linear time, and we show how to use it in Algorithm 1. In our experiments, we take the transposition cost to be 0.01, the outgoing packet deletion cost to be 4, and the incoming packet deletion cost to be 1.

`Wa-FLev` reduces the time complexity from quadratic to linear in terms of the packet sequence sizes. However, we emphasize that `Wa-FLev` has a lower accuracy than `Wa-OSAD` and `Ca-OSAD`; its accuracy is comparable to that of `Pa-FeaturesSVM`, as we will show in Section 3.4.1. Despite its lower accuracy, `Wa-FLev` was used by Juarez et al. [JAA+14] in their work to demonstrate the ineffectiveness of website fingerprinting attacks in practice.

3.1.13 k-Nearest Neighbours (Wa-kNN)

To tackle the open-world scenario, we designed `Wa-kNN` [WCN+14] (2014). `Wa-kNN` uses the k-Nearest Neighbours (k-NN) classifier, a simple supervised machine learning algorithm. k-NN starts with a set of training points $S_{train} = \{P_1, P_2, \ldots\}$, each point being a packet sequence. Let the class of P_i be $c(P_i)$. Given a testing point $P_{test} \in S_{test}$, the classifier computes the distance $d(P_{test}, P_{train})$ for each $P_{train} \in S_{train}$. The algorithm then classifies P_{test} based on the classes of the k closest training points to P_{test}.

Table 3.1: Full feature set of Wa-kNN. For the sake of efficiency, when dealing with Tor cell sequences (for which the cell length is always 512), we do not use the 3001 features in the second row. Then, we will have only 1225 features, which speeds up both feature extraction and k-NN classification.

Number of features	Description
4	$\|P\|, \|P_+\|, \|P_-\|, T_{\|P\|}$.
3001	A list M, such that $M_i = 1$ if $\exists\, i \in P_\ell$ and $M_i = 0$ otherwise. M is similar to P_U, the unique packet length feature. Given that the MTU is 1500, we have $\|M_i\| = 3001$ for all lengths between -1500 and 1500.
500	Position of the first 500 outgoing packets, i, such that $P_{\ell i} > 0$.
500	Difference in position between the first 500 outgoing packets and the next outgoing packet, $i - j$, such that $P_{\ell i} > 0$ and $P_{\ell j} > 0$.
100	Dividing P into non-overlapping windows of size 30, the number of outgoing packets in each of the first 100 windows.
3	Define a burst as a sequence of packets in the same direction (much like `Dy-VNG` [DCRS12]), and the burst length as the number of said packets. We extract the longest burst, the number of bursts, and the mean size of each burst.
6	The number of bursts with lengths longer than 2, 5, 10, 15, 20, and 50 respectively.
100	The length of the first 100 bursts.
10	The direction of the first 10 packets.
2	The mean and standard deviation of the interpacket times.

k-NN uses a distance metric d to describe how similar two packet sequences are. We want the distance to be accurate on simple encrypted data without extra padding, but also accurate if the client applies defenses that remove features from our available feature set (for example, Tor removes unique packet lengths). We therefore start with a large feature set $F = \{f_1, f_2, \ldots\}$. Each feature is a function f which takes in a packet sequence P and computes $f(P)$, a non-negative number. Conceptually, we select each feature such that packet sequences from the same page are more likely to have similar features than packet sequences from different pages. We list our full feature set of 4226 features in Table 3.1.

The distance between P and P' is:

$$d(P, P') = \sum_{i=1}^{|F|} w_i |f_i(P) - f_i(P')|$$

We learn the weights $W = \{w_1, w_2, \ldots, w_{|F|}\}$ using our new weight learning process we call WLLCC (Weight Learning by Locally Collapsing Classes). With WLLCC, we reduce weights for uninformative features. As weight learning proceeds, the k-NN distance comes to focus on features that are useful for classification. We describe WLLCC in detail in Section 3.2.5.

Despite its simplicity, k-NN has a number of advantages over other classifiers:

1. Short training and testing time. The training time consists of learning the weights W; we will show in Section 3.4.4 that this is fast. Since the distance comparison between two packet sequences is very simple, the testing time is short as well. We investigate the training and testing time further in Section 3.4.2.

2. Multi-modal classification. `Wa-kNN` is based on the important observation that a class representing a web page is *multi-modal*. A multi-modal class is a class that naturally consists of several subclasses, where points in each subclass are similar to each other, but points in different subclasses are not similar. Web pages are often multi-modal: for example, a single web page can have different localization versions for different users. k-NN is well-suited for multi-modal classes, for which different modes of the class do not need to have any relationship with each other, because only the k closest neighbours are relevant for classification. As long as the size of each mode is larger than k, k-NN will not suffer from the presence of other modes in the same class. In contrast, many other classifiers (SVM, Naïve Bayes, etc.) will take all points in the class in consideration, including those from other modes.

There are a number of ways to deal with the case when the k neighbours of a testing point differ in class assignment. Most commonly, the k neighbours act as voters in a majority scheme, possibly with their votes weighted by their distance to the testing point. In `Wa-kNN`, if any of the k neighbours disagree, we assign the testing point to the non-monitored class, even if all of the k neighbours pointed to monitored pages (but different ones). This approach gives us a strategic advantage: we can use k as a parameter to trade off TPR and FPR. A higher k means that we will more often decide that pages are non-monitored. We can also use $|C_0|$, the size of the non-monitored class, to trade off TPR and FPR. We will investigate these parameters in Section 3.4.4.

Table 3.2: Brief classification of attack strategies for different attacks. "Distance" describes whether or not the attacker computes an explicit distance for classification, or uses some other method of classification. "Feature" describes whether or not the attack operates on extracted features rather than the packet sequence itself.

Attack	Distance	Feature	Classifier
Sh-Timing	×	×	Cross-correlation score
Bi-Intertiming	×	×	Cross-correlation score
Li-Jac	×	×	Jaccard coefficient score
Li-NBayes	×	×	Naïve Bayes
He-MNBayes	×	×	Naïve Bayes
Pa-FeaturesSVM	√	√	SVM
Lu-Lev	√	×	Levenshtein score
Dy-VNG	×	√	Naïve Bayes
Ca-OSAD	√	×	SVM
Wa-OSAD	√	×	SVM
Wa-FLev	√	×	SVM
Wa-kNN	√	√	k-NN

3.1.14 Overview

In Table 3.2 we give a brief overview of the strategy of each known attack. We classify each attack according to its classification approach:

- Distance: Whether or not the attack uses a distance metric to describe packet sequence similarity. The attack may use a distance metric directly between packet sequences, or indirectly between their feature sets. If the attacker does not use an explicit distance to classify, he could do so by analyzing probabilities or some scoring mechanism (such as cross-correlation). SVM and k-NN operate on distances; Naïve Bayes operates on probabilities.
- Feature: Whether or not the attack uses a feature set. If the attack does not use a feature set, it operates directly on the packet sequence. Our golden standard for determining whether or not an attack uses a feature set is if one feature set can be replaced with another without fundamentally changing the classification approach. For example, we can exchange the feature sets in `Pa-FeaturesSVM` and `Wa-kNN`. On the other hand, while `Li-Jac` uses unique packet lengths, we do not consider it a feature set because it cannot be trivially replaced with some other feature set while continuing to use the Jaccard coefficient.

We use a simple example to illustrate how we classified these attacks. `Wa-kNN` uses distance metric learning to learn a weighted L^p-norm distance (more in Section 3.2.2). It extracts explicit features, including total packet length, sequence length and packet bursts. Therefore, it is classified as both Distance and Feature in Table 3.2.

This categorization is useful because one distance metric can be replaced with another distance metric, and one feature set can be replaced with another feature set. Attacks using those are therefore, in some sense, comparable with each other; lessons learned from one are often applied to the next to improve WF attacks.

From Table 3.2, we observe that there is a shifting trend towards distance-based and feature-based classification algorithms, although this does not constitute evidence that these methods are necessarily superior for website fingerprinting. In Section 3.2 and Section 3.3, we will specifically examine those two approaches to classification.

3.2 Classification with Distances

In Table 3.2, we saw that a number of website fingerprinting attacks compute explicit distances between packet sequences. A larger distance signifies less similarity, and vice-versa. An *informative distance* is small between packet sequences from the same web page, and large between packet sequences from different web pages. Using an informative distance is very helpful for some classifiers, including k-NN and SVM.

In this section, we will define and describe three main ways of computing distances in website fingerprinting:

1. Edit distance (Section 3.2.1). Edit distances are often used in string matching [BM00]. Researchers use edit distances for website fingerprinting due to the observation that packet sequence variation due to random network conditions is similar to string variation due to human error. All known attacks compute edit distances on packet length sequences P_ℓ, so they ignore interpacket timing.

2. L^p-norm distance (Section 3.2.2). L^p-norm distances are useful for classification. For example, k-NN often uses the Euclidean distance, which is the L^2-norm. `Wa-kNN` uses a weighted version of the L^1-norm; we learn the weights with our new distance metric learning scheme, WLLCC.

3. Kernel methods (Section 3.2.3). SVM uses kernel methods to elevate points to higher dimensions, which allows SVM to effectively find high-dimensional separators. Without kernel methods or a custom kernel, SVM would only be able to find linear separators.

We follow up by describing WLLCC in Section 3.2.4 and Section 3.2.5. WLLCC enjoys several good properties; its use in `Wa-kNN` gives us the best known attack for website fingerprinting.

3.2.1 Edit Distance

Given strings of characters A and B, an edit distance $d(A, B)$ describes how similar those strings are to each other. Edit distances are used to match strings in a wide range of applications, for example in spell checkers [BM00] and self-correcting binary codes [Lev66]. A number of website fingerprinting works have noted that edit distances between strings are analogous to differences between packet sequences [LCC10, CZJJ12, WG13b]. In the case of known website fingerprinting attacks, the alphabet is the set of all packet sizes, which does not include interpacket timing.

3.2.1.1 Levenshtein

A popular edit distance is the Levenshtein distance d_{lev} proposed by Levenshtein for self-correcting binary codes [Lev66], which is defined as the minimum number of insertions, deletions, and substitutions required to transform one string into another. In website fingerprinting, `Lu-Lev` uses the Levenshtein distance. With $A_{[:i]}$ as the prefix of A containing its first i elements, we write the shorthand $d_{i,j} = d_{lev}(A_{[:i]}, B_{[:j]})$. With $i, j > 0$, we can write the edit distance in a recursive manner:

$$d_{lev}(A_{[:i]}, B_{[:j]}) = \begin{cases} d_{i-1,j-1} & \text{if} \quad A_i = B_j \\ \min \begin{cases} d_{i-1,j} + 1 & \text{(insertion into A)} \\ d_{i,j-1} + 1 & \text{(deletion from A)} \\ d_{i-1,j-1} + 1 & \text{(substitution)} \end{cases} & \text{if} \quad A_i \neq B_j \end{cases}$$

The base case includes an empty string: $d_{lev}(A_{[:i]}, \emptyset) = i$. The above formulation gives us an $O(|A| \cdot |B|)$ algorithm to calculate the Levenshtein distance by computing all values $d_{i,j}$ while increasing i and j up to $|A|$ and $|B|$. The Levenshtein distance is a proper metric; in particular, we can prove that the triangle inequality holds by noting that the sequence of operations used to transform A into B, appended with the sequence of operations used to transform B into C, would in fact transform A into C, so that the shortest sequence of operations to transform A and C must be no longer than the sum of those to transform A into B and B into C.

If the lengths of A and B are very different, the Levenshtein distance may be very close to $|A|-|B|$: B is simply padded to A with insertions. This is often the case in website fingerprinting for sequences belonging to different classes.

The Levenshtein distance can tolerate variable insertion, deletion, and substitution costs, denoted as c_{ins}, c_{del}, c_{sub}, by changing the recursion of $d_{i,j}$ when $a_i \neq b_j$ to:

$$d_{i,j} = \min\{d_{i-1,j} + c_{ins}, d_{i,j-1} + c_{del}, d_{i-1,j-1} + c_{sub}\}$$

c_{ins} must be equal to c_{del} for the Levenshtein distance to remain a proper metric; otherwise, it loses symmetry.

3.2.1.2 Optimal String Alignment Distance

A variation of the Levenshtein distance, known as the Optimal String Alignment Distance (OSAD) d_{OSAD}, also allows restricted transpositions between adjacent elements. In website fingerprinting, `Ca-OSAD` and `Wa-OSAD` use OSAD. The Optimal String Alignment Distance changes the recursion of $d_{i,j}$ when $a_i \neq b_j$, but $a_{i-1} = b_j$ and $b_{j-1} = a_i$, to:

$$d_{i,j} = \min\{d_{i-1,j} + c_{ins}, d_{i,j-1} + c_{del}, d_{i-1,j-1} + c_{sub}, d_{i-2,j-2} + c_{trans}\}$$

Specifically, we add a transposition term when we can transpose the last two elements.

Upon careful examination of the above formula, we can see that OSAD does not allow all possible transpositions. We can interpret the restricted transpositions in OSAD between strings A and B as follows. We find the minimum number of insertions, deletions, and substitutions to string A, so that it becomes similar to string B, with the exception that some adjacent characters are swapped around. Then, we use transpositions to swap those adjacent characters around. This way, no single element will undergo two transpositions. OSAD is a proper distance metric if $c_{ins} = c_{del} = c_{sub} = c_{trans} = 1$, but possibly not otherwise:

Example 1. Set strings $A = 011$, $B = 101$, $C = 110$. We let $c_{ins} = c_{del} = c_{sub} = 1$. As in `Ca-OSAD`, we set $c_{trans} = 0.05$. Then $d_{OSAD}(A, B) = 0.05$ and $d_{OSAD}(B, C) = 0.05$, but $d_{OSAD}(A, C) = 2$, because we cannot transpose the same character twice.

`Ca-OSAD` and `Wa-OSAD` divide $d_{OSAD}(A, B)$ by $\max(|A|, |B|)$ to further normalize the OSAD. `Wa-OSAD` further changes the computation of OSAD by changing all operation costs; we investigate these changes further in Section 3.4.3.

3.2.1.3 Damerau-Levenshtein

When transpositions to transform one string into another are not restricted, i.e. transpositions between any two elements are allowed, the minimum number of insertions, deletions, substitutions,

and transpositions to transform one string into another is known as the Damerau-Levenshtein distance, first proposed by Damerau [Dam64] to correct typing errors. The algorithm to do so is similar to OSAD, except that instead of only looking at $d_{i-2,j-2}$ for transpositions, we remember the latest location of all characters, and allow a series of continuous transpositions of adjacent elements to interchange two elements that are not adjacent.

We also tested the Damerau-Levenshtein distance as a modification to `Ca-OSAD`, which uses the OSAD. [WG13b] We do not present these results in our thesis because the Damerau-Levenshtein distance performed poorly in our experiments.

3.2.2 L^p-norm Feature Distance

With explicit feature extraction of packet sequence features, we can compute the distance between packet sequences as the distance between their feature sets. `Pa-FeaturesSVM` and `Wa-kNN` use feature distances. On feature sets $F^1 = (f_1^1, f_2^1, \ldots, f_n^1)$ and $F^2 = (f_1^2, f_2^2, \ldots, f_n^2)$, the L^p-norm is:

$$d(F^1, F^2) = (\sum_{i=1}^{n} |f_i^1 - f_i^2|^p)^{1/p}$$

Sometimes, it would be convenient to allow the feature set to have variable length. To do so, we allow some of the features to take a null value. For example, the feature set may include the number of packets before each of the first 300 outgoing packets, but the packet sequence itself may not have 300 outgoing packets: we set the remainder to null. We define $|f_i^1 - f_i^2|$ to be 0 if at least one of f_i^1 and f_i^2 is null.

The above definition may perform poorly with bad features. For example, if one feature is a very large, randomly selected number, it will overwhelm the feature distance and the feature distance itself will become random, even if all other features are highly informative. In `Wa-kNN`, we attempt to nullify bad features as follows. Instead of the L^p-norm above, we use a weighted L^p-norm:

$$d(F^1, F^2) = (\sum_{i=1}^{n} w_i |f_i^1 - f_i^2|^p)^{1/p}$$

We can weight important features more so that they contribute more to the L^p norm. In the case of a bad feature, we can set w_i to 0 to discard it. To derive useful weights w_i, we present a new distance metric learning scheme, WLLCC, that has desirable properties. We defer to Section 3.2.4 for a full treatment of distance metric learning.

One major advantage of using a feature distance over an edit distance is that the results can inform us whether or not each feature was useful. Indeed, using feature distance, we find that

features relating to bursts of packets are highly useful for website fingerprinting. Furthermore, edit distances do not use timing, but feature distances do not have this restriction.

3.2.3 Kernel Method

Some classifiers, such as SVMs and kernel perceptrons, use a kernel function $K(x, y)$ to compute the distance between points x and y. Kernel functions implicitly or explicitly elevate points to a higher-dimensional space. SVM, for example, finds a linear hyperplane that separates the input data points: the linearity may be too restrictive if the number of dimensions is low. Popular kernel functions [CL11] include:

1. Polynomial of degree p: $K(x, y) = (\gamma x \cdot y + c)^p$
2. Radial basis function: $K(x, y) = e^{\gamma|x-y|^2}$
3. Sigmoid: $\tanh \gamma x \cdot y + c$

Selecting the correct kernel as well as the correct parameters is a difficult problem and there is significant work in this area [CM04].

Example 2. We can illustrate the dimension-elevating effect of the kernel method with the following example. Suppose an SVM wants to distinguish between two-dimensional points in two classes C_1 and C_2. Points from C_1 lie evenly on $x_1^2 + x_2^2 = 1$ and points from C_2 lie evenly on $x_1^2 + x_2^2 = 3$. The two classes are two concentric circles of points. Alternatively, we can write the two circles as $|x|^2 = 1$ and $|x|^2 = 3$. We can write any linear separator as $w \cdot x + b = w_1 x_1 + w_2 x_2 + b = 0$, and it is quite easy to see the poor performance of any linear separator between two circles. For example, any linear separator that guesses at least half of the points in C_2 as class 2 must guess at least half of the points in C_1 as class 2.

Now, we replace the dot product $w \cdot x$ with the radial basis function $K(w, x) = e^{|w-x|^2}$. Then with $w = \langle 0, 0 \rangle$ and $b = -e^2$, the separator becomes $e^{|x|^2} - e^2$, which does indeed separate the two circles. Members of $|x|^2 = 1$ give a negative value $e - e^2$ and members of $|x|^2 = 3$ give a positive value $e^3 - e^2$.

We note that the kernel method can be applied directly on raw data points, not only on feature sets. One can view feature extraction as effectively extracting higher-dimensional data as well: continuing Example 2, a feature defined as $x_1^2 + x_2^2$ would separate the two circles. The kernel method can therefore be considered an alternative to feature extraction. `Pa-FeaturesSVM` uses SVM with the radial basis kernel.

3.2.4 Distance Metric Learning

In distance metric learning, the learner takes as input a labeled data set S and derives a distance metric d that effectively incorporates the labels, such that elements from the same class have small d and elements from different classes have large d. Distance metric learning is useful for a wide range of applications, including k-means clustering, k-NN, and SVMs. In our work on `Wa-kNN`, we derive a practical new distance metric learning that has several good properties.

Several works on distance metric learning have focused on learning Mahalanobis metrics (i.e. Gaussian distance) [XJRN02, WBS05, GR05], where the distance can be written as:

$$d(f_1, f_2) = (f_1 - f_2)^T M (f_1 - f_2)$$

M is a positive semi-definite matrix so that d would satisfy non-negativity and the triangle inequality.

The objective function has varied in previous work, depending on the application. For example, Xing et al. minimize the same-class distance. They derive $d(f_i, f_j)$ for two feature sets with the following minimization problem:

$$\begin{aligned} \min_{M} \quad & \sum_{c(f_i)=c(f_j)} d(f_i, f_j) \\ \text{s.t.} \quad & \sum_{c(f_i)\neq c(f_j)} d(f_i, f_j) \geq 1 \end{aligned}$$

Xing et al. showed that the above optimization problem when M is diagonal can be effectively solved by applying the Newton-Raphson method [XJRN02].

Globerson and Roweis proposed Metric Learning by Collapsing Classes (MLCC). [GR05] They define a conditional distribution p, where $p(f_j|f_i)$ is the probability of picking point f_j as a neighbour for point f_i:

$$p(f_j|f_i) = \frac{e^{-d(f_i,f_j)}}{\sum_{k\neq i} e^{-d(f_i,f_k)}}$$

Ideally, we hope that $d(f_i, f_j) = 0$ ($p(f_j|f_i)$ approaches 1) when $c(f_i) = c(f_j)$, and $d(f_i, f_j)$ is very large ($p(f_j|f_i)$ approaches 0) when $c(f_i) \neq c(f_j)$. In other words, the probability of picking neighbours from the same class approaches 1 and the probability of picking neighbours from other classes approaches 0. The advantage of this formulation is that the result is a convex optimization problem in M.

Other works on distance metric learning define different optimization problems; we recommend the work by Weinberger et al. [WBS05] for a review of this field. It is not clear if different

optimization problems can be compared directly. However, we can investigate the properties induced by the optimization strategy. Weinberger et al. give several important properties of a good distance:

1. Multi-modality. A class could contain several natural subclasses of points (modes). Points in the same mode are similar, but points from different modes are not similar. For example, in website fingerprinting, a web page may have different localization versions. Packet sequences from the same localization version are more similar. A good distance should allow for multi-modality. Weinberger et al. find that a number of previous works [XJRN02, GR05] do not work well with multi-modal classes. This diminishes an advantage of using the k-NN classifier: it operates well with multi-modal classes as argued in Section 3.1.13.

2. No local minima. Convex optimization problems are advantageous, because they have only one local minimum—the global minimum. A good distance avoids local minima, and thus avoids suboptimal results.

3. Multi-class classification. Some distance metric learning algorithms work well only for binary classes. A good distance allows the classifier to classify multiple classes effectively, which is especially useful for website fingerprinting.

We will show a new distance metric learning algorithm that works well with multi-modal classes. Our algorithm collapses classes like MLCC; however, it has no global objective function, but rather performs distance metric learning based on local neighbourhoods, so we call our algorithm Weight Learning by Locally Collapsing Classes (WLLCC).

We consider weighted L^p-norm metrics instead of Mahalanobis metrics:

$$d(f_1, f_2) = (\sum_{i=1}^{n} w_i |f_{1_i} - f_{2_i}|^p)^{1/p}$$

The intersection between the L^p-norm and the Mahalanobis metrics lies at $p = 2$ and when the matrix M is diagonal.

3.2.5 Weight Learning by Locally Collapsing Classes

In Section 3.2.4 we surveyed previous several distance metric learning algorithms, and in this section we describe our new distance metric learning algorithm, Weight Learning by Locally

Collapsing Classes (WLLCC). WLLCC allows `Wa-kNN` to operate at high accuracy. Recall that we use the L^1-norm distance for `Wa-kNN`:

$$d(F^1, F^2) = (\sum_{i=1}^{n} w_i |f_i^1 - f_i^2|)$$

We chose the L^1-norm instead of any other L^p-norm because we found no major difference in accuracy when varying p, and the L^1-norm is fastest for computation.

We define d_{f_i} as follows:

$$d_{f_i}(F^1, F^2) = w_i |f_i^1 - f_i^2|$$

Thus, d can be considered the sum of d_{f_i}.

Using labeled data points in a weight training set S_{train}, we learn $W = \langle w_1, w_2, \ldots, w_{|F|} \rangle$, the weights with which we compute our feature distance. $|F|$ is the number of features in the feature set; if different points have different feature set sizes, we pad them with null features (as described in Section 3.2.2), so that $|F|$ is the same for all data points. We will also present some results on WLLCC using our website fingerprinting data set.

WLLCC is iterative and depends on local data points. It lasts for R rounds; we will see how the choice of R affects the accuracy in Section 3.4.4. For each round, we focus on a point $P_{train} \in S_{train}$, performing two steps:

1. Weight recommendation. We use the neighbours of P_{train} to determine which features are useful and which are not.

2. Weight adjustment. Based on the above, we change the weights of each feature.

Weight recommendation. The objective of the weight recommendation step is to pinpoint the weights that we want to reduce. During the weight recommendation step, the distances between P_{train} and all other $P' \in S_{train}$ are computed. We then take the closest k_{reco} points (for a parameter k_{reco}) within the same class $S_{good} = \{P_1, P_2, \ldots, P_{k_{reco}}\}$ and the closest k_{reco} points within all other classes $S_{bad} = \{P'_1, P'_2, \ldots, P'_{k_{reco}}\}$.[4]

Let us denote

$$d_{maxgood_i} = \max\{P \in S_{good} : d_{f_i}(P_{train}, P)\}$$

In other words, $d_{maxgood_i}$ is the maximum distance d_{f_i} between P_{train} and all points in S_{good}.

[4]Note that k_{reco} does not have to be the same as the number of neighbours used for classification in k-NN; in our implementation k_{reco} is always set at 5, and we did not observe significant accuracy changes when varying k_{reco}.

For each feature, we compute the number of relevant bad distances, n_{bad_i}, as

$$n_{bad_i} = |\{P' \in S_{bad} : d_{f_i}(P_{train}, P') \leq d_{maxgood_i}\}|$$

n_{bad_i} indicates the usefulness of feature f_i in helping to distinguish S_{bad} from S_{good}. A large value of n_{bad_i} means that feature f_i is not useful at distinguishing members of P_{train}'s class from members of other classes, and so WLLCC decreases the weight of f_i; for example, features perfectly covered by a defence (such as unique packet lengths in Tor) will always have $n_{bad_i} = k_{reco}$, its maximum possible value. Conversely, small values of n_{bad_i} indicate that feature f_i is helpful and WLLCC should increase its weight. We examine how to adjust weights next.

Weight adjustment. We adjust the weights to keep $d(P_{train}, D_{bad})$ the same while reducing $d(P_{train}, S_{good})$. For each i such that $n_{bad_i} \neq \min\{n_{bad_1}, n_{bad_2}, \ldots, n_{bad_{|F|}}\}$, we reduce the weight by $\Delta w_i = w_i \cdot 0.01$. We then increase all weights w_i with

$$n_{bad_i} = \min\{n_{bad_1}, n_{bad_2}, \ldots, n_{bad_{|F|}}\}$$

We increase weights equally such that $d(P_{train}, S_{bad})$ remains the same. We keep $d(P_{train}, S_{bad})$ the same rather than increasing it because increasing the distance may push bad points towards other bad neighbours, which may disrupt convergence towards optimality; this strategy is also used in previous work [WBS05].

We achieved our best results with two more changes to the way we reduce weights, as follows:

- We further multiply $\Delta w_i = w_i \cdot 0.01$ by n_{bad_i}/k_{reco}. Therefore, a weight with greater n_{bad_i} (a less informative weight) will be reduced more.
- We decrease Δw_i if P_{train} is already well classified. We define N_{bad} as:

  $$N_{bad} = |\{P' \in S_{bad} : d(P_{train}, P') \leq d_{maxgood}\}|$$

  Specifically, we multiply Δw_i by $(1 + N_{bad})/k_{reco}$. N_{bad} measures how poorly the current point is classified. Points which are already well classified (low N_{bad}) will have a small effect on weight adjustment (low Δw_i) in each iteration. We add 1 to ensure that even perfectly classified points still have some small impact on the weights; we would not want weight adjustment to nullify their perfect classification.

As described in Section 3.2.4, avoidance of local minima is a desirable property, but WLLCC operates only locally with no global objective function, so it may reach local minima during optimization. To avoid local minima, we can add randomness to the weight vector, which gives us a chance of finding better solutions than a deterministic algorithm. We will test a method for adding randomness to WLLCC in Section 3.4.4.

3.3 Classification with Features

In Section 3.2, we showed that we can derive distances by extracting features from packet sequences. For example, `Dy-VNG`, `Wa-kNN`, and `Pa-FeaturesSVM` extract features to classify packet sequences. Other attacks such as `Li-Jac` and `Li-NBayes` do not explicitly extract feature sets, but nevertheless their classification is dependent on observing specific features in packet sequences.

Classification is effective only if the feature set is effective for the problem. Random, uninformative features may negatively impact classification, even if we attempt to mollify such an effect with distance metric learning. In this aspect, we require expert knowledge to choose features correctly.

In this section, we will analyze features used in previous WF attacks. We propose a categorization scheme in Section 3.3.1 to divide features into four categories: unique packet lengths, packet length frequency, packet ordering, and interpacket timing. We will then attempt to illustrate how important each set of features is to each WF attack in Section 3.3.2. To do so, we generate two sets of packet sequences, between which only one feature differs, and analyze the accuracy of the classifier in attempting to these two sets.

3.3.1 Feature Categorization

In this section we categorize website fingerprinting features into four categories: unique packet lengths, packet length frequency, packet ordering, and interpacket timing. We will then proceed by proving that this categorization of features has two desirable properties:

1. Complete. If two packet sequences differ, they must differ in at least one of the feature categories.
2. Mutually independent. For any packet sequence P and each of the four categories, we can construct P' such that P differs from P' only in that feature category.

We will see that having a complete and mutually independent categorization of features allows us to derive interesting observations about known website fingerprinting attacks.

Unique packet lengths. Packet lengths are a powerful feature for website fingerprinting. GET request lengths are partly determined by the length of the resource name. Incoming packets are

almost always sent at the Maximum Transmission Unit (MTU), with the length of the last packet indicating the size of the resource (modulo the MTU).

Most packet lengths of a page are unlikely to change unless resource lengths change. WF attacks almost always consider packet lengths unless they are designed for the Tor scenario, which uses fixed-size cells.

We define P_U as the sequence of unique packet lengths of P, ordered in terms of size. In other words, P_U has no duplicates, and:

$$\ell \in P_U \iff \ell \in P_\ell$$

We require that packet sequences must have at least one outgoing packet and one incoming packet, because the client needs to send a request to load a page, and a reply must come back. Therefore we have $|P_U| \geq 2$. Then we say that sequences P and P' have different unique packet lengths if:

$$P_U \neq P'_U$$

Note that if we are examining Tor cell sequences, then we write $P_U = \langle -1, 1 \rangle$, since all cells have the same size.

Sequence lengths. We split the sequence length $|P|$ into outgoing and incoming packets. We count the number of packets in each direction, denoted as $|P_+|$ and $|P_-|$, and we write $L_P = \langle |P_+|, |P_-| \rangle$. We say that two packet sequences P and P' have different sequence lengths if:

$$L_P \neq L_{P'}$$

The total incoming sequence length is roughly the size of the page, which is a consistently useful feature, as it may reflect the content density and type (video, images, text) of the page. The total size may however change due to random advertisements and updated content. The total outgoing sequence length represents the number of requests necessary to load the page.

Packet ordering. The structure of a page induces a logical order in its packet sequence. For example, a client sends a `GET` request for a resource after learning that it is necessary to load the page from incoming data. An attacker may be able to infer information about the content of each packet by observing packet ordering. Packet ordering depends on network conditions: it may vary due to bandwidth and latency. Tor developers have implemented a prototype defense based on packet ordering by randomizing request order; we describe this defense in detail when analyzing WF defenses in Section 5.2.1.5.

Let us define a lookup function u_P for the unique packet length sequence P_U such that $u_P(\ell)$ returns the unique index of P_U where ℓ is located, i.e. $P_{U\,u_P(\ell)} = \ell$. We rewrite packet sequence P_ℓ as P_O, such that:

$$P_{O_i} = u_P(P_{\ell_i})$$

In other words, P_O is like P_ℓ, but instead of packet lengths, each element is an index of P_U. Then, we say that two packet sequences P and P' with $|P'| \geq |P|$ have different ordering if:

$$\exists i, 1 \leq i \leq |P| \text{ such that } P_{O_i} \neq P'_{O_i}$$

Interpacket timing. Interpacket times often reveal the logical relationship between packets. For example, viewing from the client's end, the outgoing server connection SYN and the incoming SYN-ACK will differ by a round-trip time; so will the GET request and the first packet of that resource. If the attacker is near the client, then the attacker can infer that an outgoing packet and an incoming packet cannot be logically dependent if their interpacket time is less than one RTT.

We say that P and P' with $|P'| \geq |P|$ have different interpacket timings if:

$$\exists i, 1 \leq i \leq |P| \text{ such that } P_{t_i} \neq P'_{t_i}$$

Example 3. Consider the packet sequence:

$$P = \langle (0, 537), (0.23, -814), (0.01, 537), (0.77, -1500), (0.00, -1500), (0.01, -271) \rangle$$

Dividing into packet lengths and interpacket times, we have

$$\begin{aligned} P_\ell &= \langle 537, -814, 537, -1500, -1500, -271 \rangle \\ P_t &= \langle 0, 0.23, 0.01, 0.77, 0.00, 0.01 \rangle \end{aligned}$$

In terms of our first three feature categories, we have

$$\begin{aligned} P_U &= \langle -1500, -814, -271, 537 \rangle \\ L_P &= \langle 2, 4 \rangle \\ P_O &= \langle 4, 2, 4, 1, 1, 3 \rangle \end{aligned}$$

First, we show that our feature categorization is complete:

Lemma 1. If $P \neq P'$, then they must differ in at least one of the four features above.

Proof. We assume $P \neq P'$, and yet P and P' are equal in all four features, and show a contradiction. First, we note that $\sum L_P = |P_+| + |P_-| = |P|$. If $|P| \neq |P'|$ then $L_P \neq L_{P'}$, so sequence lengths are different. We therefore proceed with $|P| = |P'|$ and $P \neq P'$.

For interpacket timing, if $P_t \neq P'_t$ but $|P_t| = |P'_t|$, then $\exists i$ such that $P_t(i) \neq P'_t(i)$, so they have different interpacket timings according to the definition.

So we have $P_t = P'_t$, yet $P \neq P'$; thus they must differ in packet lengths. We proceed with $P_\ell \neq P'_\ell$ and $|P_\ell| = |P'_\ell|$, so for some i we have $P_{\ell_i} \neq P'_{\ell_i}$. Suppose they have the same unique packet lengths, we seek to prove that they must then have different packet ordering. With $P_U = P'_U$, and suppose $P_{U_j} = P_{\ell_i}$ and $P_{U_{j'}} = P'_{\ell_i}$. We must then have $j \neq j'$ as P_U has no repeating elements. Looking at the packet ordering P_O and P'_O, recall that the definition of P_O is

$$\begin{aligned} P_{O_i} &= u_P(P_{\ell_i}) &= j \\ P'_{O_i} &= u_{P'}(P'_{\ell_i}) &= j' \end{aligned}$$

Therefore $P_O \neq P'_O$; the two sequences P and P' have different packet ordering. It is therefore impossible that $P_\ell \neq P'_\ell$ and yet P and P' have the same unique packet lengths, sequence length, and packet ordering. □

Our feature categorization is also mutually independent:

Lemma 2. Given any packet sequence P and any subset of the above features, there exists a packet sequence P' such that P and P' only differ in this subset of features.

Proof. The proof is trivial for timing; an operation to substitute any interpacket time with another will change P_t without changing P_ℓ. We focus on P_ℓ for the following. For each of the three feature categories—unique packet lengths, sequence length, and packet ordering—we prove mutual independence by constructing a P'_ℓ such that P'_ℓ differs from P_ℓ only in one feature category.

For unique packet lengths, let ℓ_{max} be the largest outgoing unique packet length. Consider a packet padding operation that changes all packets in P_ℓ of size ℓ_{max} into packets of size ℓ'_{max} with $\ell'_{max} > \ell_{max}$. That is to say:

$$\begin{aligned} P'_{\ell_i} &= P_{\ell_i} && \text{if } P_{\ell i} \neq \ell_{max} \\ P'_{\ell_i} &= \ell'_{max} && \text{if } P_{\ell i} = \ell_{max} \end{aligned}$$

Since both $\ell'_{max} > 0$ and $\ell_{max} > 0$ (they are both outgoing packets), $L_{P'} = L_P$; the number of incoming and outgoing packets has not changed. Similarly $P_O = P'_O$ as well: for all i such that $P_{\ell_i} = \ell_{max}$,

$$P'_{O_i} = |P'_U| = |P_U| = P_{O_i}$$

So this operation only changes unique packet lengths.

For sequence length, we consider a packet appending operation, where we append a packet of size $\ell_{add} > 0$ to P_ℓ to construct P'_ℓ, with the stipulation that ℓ_{add} is already in P_U: $P_{U_i} = \ell_{add}$. This means that $|P'_{\ell+}| = |P_{\ell+}| + 1$. $P_U = P'_U$; ℓ_{add} is in both of them. Also note that from the definition of packet ordering, we only examine the order of the shorter of the two sequences. Since P'_ℓ is P_ℓ with one extra packet at the end, they have the same packet ordering.

For packet ordering, we consider a transposition operation. $P'_\ell = P_\ell$ except at two locations i and j where $P_{\ell_i} \neq P_{\ell_j}$: we have $P'_{\ell_i} = P_{\ell_j}$ and $P'_{\ell_j} = P_{\ell_i}$. Such two locations must exist because we required that there is at least one incoming and one outgoing packet in each packet sequence. Since the transposition does not change the set of unique packet lengths, $P_U = P'_U$. Furthermore it does not change the sequence length, so $L_P = L_{P'}$. □

The reader can check that the above four operations can be combined with each other, such that we can always construct P' that differs from P in any two, three, or all four feature categories.

It is interesting to note that the three constructions for P'_ℓ correspond well to the three possible operations of the Damerau-Levenshtein distance (see Section 3.2.1): substitution, insertion/deletion, and transposition, in order. We will use these constructions to categorize attacks in the next section.

3.3.2 Attack Categorization

We demonstrate how we can categorize attacks with the four feature categories above. To do so, we use the four operations defined in Lemma 1. We formally define four generators, which take as input a packet sequence P, and output a packet sequence P' where only one feature category has changed. Each generator accepts a parameter $d \in [0, 1]$, which defines how much the generator changes.

Unique generator. Choose $d \cdot |P_U|$ random different packet lengths in P_U. Change all occurrences of them in P_ℓ to different packet lengths not in P_U.

Length generator. Append $d \cdot |P|$ packets to P_ℓ. The ith packet appended is the same as P_{ℓ_i}.

Order generator. Transpose $d \cdot |P|$ random pairs of packet lengths. These packet lengths must be different, though the same pairs can be transposed again.

Table 3.3: Feature categorization of `Pa-FeaturesSVM`. We write $\surd$ if the generator is likely to change the SVM feature set (much greater than 50% chance at $d = 1$); ◯ if the generator is unlikely to change the SVM feature set (much smaller than 50% chance at $d = 1$); and $\times$ if the generator never changes the SVM feature set.

SVM Feature set	Unique	Length	Order	Timing
Size markers	$\surd$	$\surd$	$\surd$	$\times$
HTML size	◯	$\times$	◯	$\times$
Number of packets	$\times$	$\surd$	$\times$	$\times$
Number markers	$\times$	$\surd$	$\surd$	$\times$
Unique packet sizes	$\surd$	$\times$	$\times$	$\times$
Percentage of incoming packets	$\times$	$\surd$	$\surd$	$\times$

Timing generator. Pick $d \cdot |P|$ times $T_i \in P_T$,[5] and randomly change them to some value between T_{i-1} and T_{i+1}.

Each generator only changes one feature category. We show two methodologies to analyze attacks and defenses using the generators. First, we do so analytically by examining the definition and code of previous attacks. We illustrate this method on `Pa-FeaturesSVM`. Table 3.3 categorizes this attack's feature sets, and the reader should refer to their work [PNZE11] for a detailed description of these feature sets. We use three symbols:

$\surd$ if the generator is likely to change the SVM feature set (much greater than 50% chance at $d = 1$);

◯ if the generator is unlikely to change the SVM feature set (much smaller than 50% chance at $d = 1$);

$\times$ if the generator never changes the SVM feature set.

We see from Table 3.3 that several of their feature sets are highly sensitive to sequence length and ordering. This would explain their greater success on Tor compared to previous work, as unique packet lengths do not occur on Tor and timing is weakened due to random circuit construction. Many of the older attacks ignore sequence length and ordering, so they fail on Tor.

One disadvantage of the above approach is that we must carefully analyze every aspect of an attack, and we can make mistakes if we misinterpret the classification approach. Generators

[5] P_T is the list of total elapsed times, not interpacket times.

ameliorate this problem by allowing us to categorize attacks with empirical testing. We scale d for each generator so that $d = 0$ represents no change and $d = 1$ represents changing a number of packets equal to $|P|$. Then, we take a random class $C_0 \subseteq S$, and apply the generator to obtain C_1 for each feature category.

We examine the accuracy of each classifier in distinguishing between C_0 and C_1. To do so, we pick 50 elements from a random monitored page in the non-Tor data set; we call these 50 elements class C_0. Then, we apply the generator with some degree d to each of those 50 elements, and label the transformed packet sequences as class C_1. The classifier attempts to distinguish between C_0 and C_1: in other words, the classifier attempts to detect the feature change induced by the generator.

We present the results in Table 3.4 on five settings: $d = 0.05$, $d = 0.1$, $d = 0.2$, $d = 0.5$, $d = 1$. A higher accuracy indicates that the classifier is more sensitive to changes in that feature category, whereas an accuracy of 0.5 indicates that the classifier is completely insensitive to that category. From this table, we can conclude several interesting facts:

- In general, sensitivity of Timing was either weak or non-existent. `Sh-Timing` and `Bi-Intertiming` showed weak signs of sensitivity. Indeed, most attacks do not take timing as input. `Wa-kNN`, in fact, uses one feature for the total duration of the packet sequence, but our Timing generator does not change the total duration.

- `Li-Jac` is an attack of absolutes: it is completely sensitive to Unique, but insensitive to all other generators. In contrast, `Pa-FeaturesSVM` takes the middle road: it is somewhat sensitive to Unique, Length, and Order at all degrees.

- The best known attacks on the non-Tor scenario (`Li-Jac`, `Li-NBayes`, `He-MNBayes`) are all completely sensitive to Unique. On the other hand, none of the edit distance attacks use packet length, but we must note that this is only because of our implementation for the Tor scenario. The original implementation of `Ca-OSAD` by Cai et al. does in fact use packet length, with different packet lengths being analogous to different letters of the alphabet in string comparison. It is possible to change all five edit distance attacks so that they are sensitive to Unique.

- All attacks are sensitive to Length with the exception of `Li-Jac`. This comes as no surprise: the sequence length is a powerful feature for classification, and it is somewhat difficult to ignore (as each attack scans through the whole list of packets to derive features or computations).

- There is a clean split of sensitivity towards Order between the older attacks (which were designed for proxies that do not use Tor) and the newer attacks starting from

Table 3.4: Accuracy of 12 WF attacks in distinguishing between two classes of packet sequences. The first class comes from a random monitored page in the non-Tor data set, and the second is generated from sequences in the first class using a Unique, Length, Order, or Timing generator with varying degree.

Generator type	Unique					Length				
Degree (d)	0.05	0.1	0.2	0.5	1	0.05	0.1	0.2	0.5	1
Sh-Timing [SW06]	0.5	0.5	0.5	0.5	0.5	0.66	0.63	0.71	0.81	0.94
Bi-Intertiming [BLJL06]	0.54	0.55	0.65	0.74	0.63	0.5	0.62	0.66	0.65	0.7
Li-Jac [LL06]	1.0	1.0	1.0	1.0	1.0	0.5	0.5	0.5	0.5	0.5
Li-NBayes [LL06]	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.98	1.0	1.0
He-MNBayes [HWF09]	1.0	1.0	1.0	1.0	1.0	0.95	0.84	0.98	0.74	0.5
Lu-Lev [LCC10]	0.5	0.5	0.5	0.5	0.5	0.72	0.87	0.84	0.98	0.94
Pa-FeaturesSVM [PNZE11]	0.8	0.73	0.81	0.83	0.77	0.63	0.73	0.74	0.84	0.74
Dy-VNG [DCRS12]	0.5	0.5	0.5	0.66	0.5	0.5	0.52	0.51	0.54	0.65
Ca-OSAD [CZJJ12]	0.5	0.5	0.5	0.5	0.5	0.8	0.94	0.9	0.83	1.0
Wa-OSAD [WG13b]	0.5	0.5	0.5	0.5	0.5	0.76	0.67	0.83	0.83	0.83
Wa-FLev [WG13b]	0.5	0.5	0.5	0.5	0.5	0.77	0.9	0.8	0.83	0.97
Wa-kNN [WCN+14]	1.0	1.0	1.0	0.99	0.98	0.93	0.87	0.99	0.89	0.99
Generator type	Order					Timing				
Degree (d)	0.05	0.1	0.2	0.5	1	0.05	0.1	0.2	0.5	1
Sh-Timing [SW06]	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.54	0.5	0.58
Bi-Intertiming [BLJL06]	0.5	0.5	0.5	0.5	0.57	0.5	0.51	0.5	0.58	0.56
Li-Jac [LL06]	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Li-NBayes [LL06]	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
He-MNBayes [HWF09]	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Lu-Lev [LCC10]	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Pa-FeaturesSVM [PNZE11]	0.57	0.69	0.67	0.67	0.78	0.5	0.5	0.5	0.5	0.5
Dy-VNG [DCRS12]	0.5	0.55	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Ca-OSAD [CZJJ12]	0.61	0.65	0.83	0.82	0.83	0.5	0.5	0.5	0.5	0.5
Wa-OSAD [WG13b]	0.64	0.75	0.83	0.83	0.83	0.5	0.5	0.5	0.5	0.5
Wa-FLev [WG13b]	0.69	0.9	0.83	0.87	0.93	0.5	0.5	0.5	0.5	0.5
Wa-kNN [WCN+14]	0.5	0.86	0.98	0.95	1.0	0.5	0.5	0.5	0.5	0.5

Pa-FeaturesSVM (which were designed for Tor). In fact, there is a gradual increase in the sensitivity towards Order.

- Averaging across all packet sequences, Wa-kNN was the most sensitive to all features, cleanly distinguishing between the original and the generated class with almost all settings except for Timing. We should certainly say, however, that there is no direct link between sensitivity and the overall classification accuracy. Wa-kNN is sensitive to packet features because we designed it to defeat website fingerprinting defenses that leave a few features uncovered.

We can also use a similar methodology to examine website fingerprinting defenses rather than attacks. To do so, we first construct two classes of packet sequences that differ only by one feature category using one of the above generators. Then, we apply the WF defense on the two classes. Finally, we apply a feature-based classifier (like SVM and k-NN) to try to distinguish between two classes. If the classifier fails, we can argue that the WF defense covers this feature. However, this approach has some problems. If the WF defense adds some small random perturbation to the packet sequence, the two classes may be distinguishable by the classifier yet the defense may be provably effective on website fingerprinting in general. The choice of the feature-based classifier would also be largely arbitrary. The conclusions drawn from this approach are therefore rather limited. In our previously published paper [CNW+14] we presented results using this approach, but we will not include the results in this work. Instead, we will examine another approach to prove the effectiveness of a defense in general, with respect to any feature set, in Section 5.1.4.

3.4 Evaluation

We described twelve known website fingerprinting attacks in Section 3.1. In this section, we evaluate them against the data sets we collected using the methodology in Section 2.4. We compare their classification accuracy in Section 3.4.1 under the open-world and closed-world models, and we compare their storage, training and testing time in Section 3.4.2. In Section 3.4.3 we specifically evaluate Wa-OSAD to validate that the changes we made to Ca-OSAD improve classification accuracy. Similarly in Section 3.4.4 we evaluate Wa-kNN and its WLLCC algorithm (see Section 3.2.4), varying the parameters of Wa-kNN and observing the effects on classification accuracy.

3.4.1 Comparison

In this section we test and compare all website fingerprinting attacks in Section 3.1 under three scenarios: closed world without Tor, closed world on Tor, and open world on Tor. As described in Section 3.1, these three scenarios correspond to an increasing progression of difficulty for website fingerprinting. The earliest works tackle the first scenario, and later works move on to the second and third scenarios. As a shorthand, we refer to a testing data set with n_{mpage} monitored pages, with n_{minst} packet sequence instances for each page visited, and n_{nmpage} non-monitored pages as $n_{mpage} \times n_{minst} + n_{nmpage}$. In the closed-world scenario ($n_{nmpage} = 0$) we do not write the added term.

We split the data into testing and training sets using 10-fold testing, such that the size of the training set is 90% of the full data set, and the rest are used for testing. The same packet sequence will never simultaneously occur as both a training and testing element. Our presented accuracy values are obtained by testing each data point in our data set once; we do not perform multiple runs as this would not change the accuracy or the standard deviation. We present our accuracy values followed by the standard deviation, which is calculated as the standard deviation of the binomial distribution:

$$\sqrt{\frac{a(1-a)}{n}}$$

In the above, a is the mean accuracy and n is the number of tests in the experiment. The binomial distribution is well-approximated by the normal distribution when: [WMMY12]

$$an > 0.5, (1-a)n > 0.5$$

We now argue that we satisfy the above condition. n, the number of tests, is equal to the total size of the testing set in our experiments. Since we test every element, n is the sum of the number of positive (n_P) and negative n_N (monitored and non-monitored) elements. Supposing that a is the TPR, $an \geq an_P$, the number of true positives; similarly $(1-a)n \geq (1-a)n_N$, the number of false negatives, which are certainly both greater than 0.5 (unless we observe a TPR of 1 or 0). We can make the same argument for the FPR. Thus, our binomial distribution is indeed close to a normal distribution. Therefore, we can use the relationship between the pdf of the normal distribution and its standard deviation here as well: one standard deviation covers about 70% of all data and two covers 95%.

We must note that our standard deviation does not reflect possible changes to the data set. Changes to the data set can be significant: some sites may be more difficult to classify than others, and Tor network conditions change frequently. Nevertheless, our results in this section are quite similar to previously published works on website fingerprinting on Tor that introduced

Table 3.5: TPR of the twelve WF attacks we have surveyed, under the closed-world scenario on the non-Tor data set with four data set sizes: 10×10, 10×100, 100×10, and 100×100.

Attack	10×10	10×100	100×10	100×100
Sh-Timing [SW06]	0.72 ± 0.04	0.75 ± 0.01	0.45 ± 0.02	0.509 ± 0.005
Bi-Intertiming [BLJL06]	0.82 ± 0.04	0.83 ± 0.01	0.61 ± 0.02	0.591 ± 0.005
Li-Jac [LL06]	1 ± 0	0.998 ± 0.001	0.985 ± 0.004	0.976 ± 0.002
Li-NBayes [LL06]	1 ± 0	0.999 ± 0.001	0.986 ± 0.004	0.993 ± 0.0008
He-MNBayes [HWF09]	1 ± 0	0.996 ± 0.002	0.991 ± 0.003	0.988 ± 0.001
Lu-Lev [LCC10]	0.98 ± 0.01	0.993 ± 0.002	0.85 ± 0.01	0.934 ± 0.002
Pa-FeaturesSVM [PNZE11]	0.65 ± 0.05	0.73 ± 0.01	0.54 ± 0.02	0.500 ± 0.005
Dy-VNG [DCRS12]	0.18 ± 0.04	0.15 ± 0.01	0.01 ± 0	0.01 ± 0
Ca-OSAD [CZJJ12]	0.97 ± 0.02	0.990 ± 0.003	0.89 ± 0.01	0.951 ± 0.002
Wa-OSAD [WG13b]	0.99 ± 0.01	0.994 ± 0.002	0.935 ± 0.008	0.973 ± 0.002
Wa-FLev [WG13b]	0.94 ± 0.02	0.941 ± 0.007	0.84 ± 0.01	0.838 ± 0.004
Wa-kNN [WCN+14]	0.99 ± 0.01	0.992 ± 0.003	0.921 ± 0.009	0.952 ± 0.002

these attacks [PNZE11, CZJJ12, WG13b, WCN+14], though we use a new data set with different sites and different network conditions.

Closed world without Tor. We apply each attack on the non-Tor data set, in the closed-world setting, under 4 scenarios: 10×10, 10×100, 100×10, and 100×100. The closed-world setting without Tor is the oldest setting upon which many of the first website fingerprinting works demonstrated their effectiveness [SSW+02, BLJL06, LL06, HWF09]. Web pages have changed significantly since then (about 10 years ago), growing much larger over time and incorporating various media in them. We want to know if this has changed the results of website fingerprinting significantly. The results are presented in Table 3.5.

We make the following observations from Table 3.5:

- `Li-Jac` and `Li-NBayes` performed surprisingly well, with an accuracy significantly higher than their published results [LL06]; `He-MNBayes` is also very accurate, matching their published results [HWF09]. Our results show no significant difference in accuracy between `Li-Jac`, `Li-NBayes` and `He-MNBayes`. It may be that one of these attacks would pull away from the others with more difficult settings (e.g. 1000×10), but we do not have such data.

Table 3.6: TPR of the twelve WF attacks we have surveyed, under the closed-world scenario on the Tor data set with four data set sizes: 10×10, 10×100, 100×10, and 100×100.

Attack	10×10	10×100	100×10	100×100
Sh-Timing [SW06]	0.42 ± 0.05	0.61 ± 0.02	0.16 ± 0.01	0.298 ± 0.005
Bi-Intertiming [BLJL06]	0.63 ± 0.05	0.69 ± 0.01	0.34 ± 0.02	0.398 ± 0.005
Li-Jac [LL06]	0.1 ± 0	0.1 ± 0	0.01 ± 0	0.01 ± 0
Li-NBayes [LL06]	0.91 ± 0.03	0.86 ± 0.01	0.37 ± 0.02	0.490 ± 0.005
He-MNBayes [HWF09]	0.11 ± 0.03	0.12 ± 0.01	0.021 ± 0.005	0.017 ± 0.001
Pa-FeaturesSVM [PNZE11]	0.88 ± 0.03	0.913 ± 0.009	0.59 ± 0.02	0.806 ± 0.004
Dy-VNG [DCRS12]	0.56 ± 0.05	0.49 ± 0.02	0.10 ± 0.01	0.089 ± 0.003
Lu-Lev [LCC10]	0.57 ± 0.05	0.55 ± 0.02	0.12 ± 0.01	0.100 ± 0.003
Ca-OSAD [CZJJ12]	0.96 ± 0.02	0.971 ± 0.005	0.86 ± 0.01	0.937 ± 0.002
Wa-OSAD [WG13b]	0.96 ± 0.02	0.98 ± 0.004	0.915 ± 0.009	0.967 ± 0.006
Wa-FLev [WG13b]	0.93 ± 0.03	0.84 ± 0.01	0.75 ± 0.01	0.784 ± 0.004
Wa-kNN [WCN+14]	0.95 ± 0.02	0.949 ± 0.007	0.86 ± 0.01	0.948 ± 0.002

- Comparing columns 1 and 2, and columns 3 and 4, we can see the effect of increasing the number of training instances on accuracy. Most attacks do not appear to benefit significantly, the largest beneficiaries being `Wa-kNN` and `Sh-Timing` each gaining about a 6% increase in accuracy. This is perhaps due to the fact that unique packet lengths appear consistently in web pages.
- Comparing columns 1 and 3, and columns 2 and 4, we can see the effect of increasing the number of monitored sites on accuracy. Increasing the number of monitored sites makes the classification problem more difficult. The loss is contained within a few percentage points for `Li-Jac`, `Li-NBayes`, `He-MNBayes` and `Wa-kNN`, while other attacks suffer more.

Closed world on Tor. For the closed-world scenario on Tor, all cells have the same fixed constant length. Further, as there is more noise due to random circuit selection and circuit construction, classification is more difficult. As before, we test the closed-world accuracy under 4 scenarios: 10×10, 10×100, 100×10, and 100×100. We present the results in Table 3.6. We draw the reader's attention to several interesting results in Table 3.6:

- Our results for the newest attacks are all quite similar to previously published results. We find that `Ca-OSAD`, `Wa-OSAD`, and `Wa-KNN` all achieve a very high TPR on Tor, while

Pa-FeaturesSVM and Wa-FLev have a somewhat lower accuracy. Wa-OSAD has the highest accuracy overall on 10×100, 100×10 and 100×100, followed closely by Wa-kNN. In our experiment, the former took 4937 CPU hours on the 100×100 data set, while the latter took 156 CPU seconds.

- Attacks based solely on unique packet lengths, including Li-Jac and He-MNBayes, do not work on Tor, of course. Since Li-Jac resorts completely to random guessing, we write its standard deviation as 0. Herrmann et al. also reported around a 2% accuracy on Tor under a similar setting, although their data set had 775 pages.

- The chief difference between Table 3.6 and Table 3.5 comes from the fact that Tor pads unique packet lengths to a fixed size. Almost all attacks perform significantly better on the non-Tor data set, especially Li-NBayes and He-MNBayes. We pad all unique lengths to the MTU for the non-Tor data set, and test Li-NBayes and He-MNBayes on them. We observe respective accuracies of 0.600 ± 0.005 and 0.024 ± 0.002 on the padded non-Tor data set, which are much closer to the results on the Tor data set. Since packet padding has a very low overhead, any privacy-enhancing technology for web browsing should certainly pad packet sizes to a fixed length. This fact implies that results on the Tor data set are *more important* for website fingerprinting in general than the non-Tor data set.

- The only attack that performs better on the Tor data set than the non-Tor data set is Pa-FeaturesSVM, which is designed specifically to perform website fingerprinting on JAP and Tor, where unique packet lengths are a covered feature. Its accuracy is much lower on the non-Tor data set, and it is comparable to Sh-Timing and Bi-Intertiming.

- There is a significant disparity between our measurement for Dy-VNG and their reported accuracy. We suspect this is because we made several significant assumptions when attempting to implement their attack, there being no explanation of certain intricacies in their work [DCRS12]. We assumed that they use a Kernel Density Estimator to compute probability (like in Li-NBayes), and we assumed that the attack learns a single KDE for all burst lengths of the same site. For example, it is possible that they used multiple KDEs for different bursts. We conclude that we cannot reproduce their results.

Open world on Tor. The closed-world scenario offers an interesting comparison between attacks, but it is not realistic for the website fingerprinting scenario, because the number of possible web pages is millions of times greater than our data set. This can be resolved by moving to the open-world scenario, where we add a non-monitored class that covers other web pages. We further restrict that the attacker's training set can never include the same non-monitored web

pages as the testing set, which means that the attacker must attempt to identify that pages he has never seen before are non-monitored. Thus, this scenario fits any number of pages outside of the monitored data set.

We present both the TPR and TNR of each scheme in Table 3.7 under four data set sizes: $10 \times 5 + 50$, $10 \times 50 + 500$, $100 \times 5 + 500$, and $100 \times 50 + 5000$. Compared to the closed-world scenario, the attacker sees fewer instances of each site, but we add an equal number of non-monitored pages to the data set. Table 3.7 shows us the following interesting results:

- `Wa-kNN` performs by far the best in the $100 \times 50 + 5000$ scenario. It has a true positive rate of 70% and a true negative rate of 98%. If the base rate of all 100 pages added together is 10%, for example, then 80% of all "positive" classifications by the classifier will be true, and the rest will be false. Other attacks are far from this level of accuracy.

- `Wa-FLev` collapsed under the weight of the non-monitored class, resorting to returning a false classification almost all the time. We suspect this is because the SVM implementation uses one-to-one testing. One-to-one testing can be considered as a tournament where each participant (class) competes against each other participant once, the winner being the class that the point more likely belongs to, and the class that won the most matches is the final classification. As the non-monitored class is significantly larger than any monitored class, the matches are heavily biased towards the non-monitored class. Curiously, `Pa-FeaturesSVM` shows no such behaviour despite also using the one-to-one SVM, possibly because it is able to adjust for larger classes using the kernel method.

- For the above reason, we set the non-monitored class size to the same size as each monitored class for `Lu-Lev`, `Ca-OSAD` and `Wa-OSAD`, anticipating that those attacks would collapse as well if we had maintained the original non-monitored class size. So the data sizes are $10 \times 5 + 5$, $10 \times 50 + 50$, $100 \times 5 + 5$, and $100 \times 50 + 50$ only for those three attacks. This results in a large standard deviation for the non-monitored page set. Still, `Ca-OSAD` and `Wa-OSAD` performed poorly in the open-world scenario, showing high false positive rates.

- Despite the fact that half of all elements are from the non-monitored class, there are four attacks that almost never identify a testing element as the non-monitored class: `Bi-Intertiming`, `Li-NBayes`, `He-MNBayes` and `Dy-VNG`. In particular, we observe that `Bi-Intertiming` and `Li-NBayes` still maintain a reasonable accuracy on the monitored class. This is because all of these attacks are feature-based attacks, learning classes by extracting features from all of its training elements. To these attacks, the large non-monitored class is merely an especially inconsistent class; the size does not affect these attacks, unlike the edit distance attacks discussed above.

Table 3.7: TPR and TNR of the twelve WF attacks we have surveyed, under the open-world scenario on the Tor data set with four data set sizes: $10 \times 5 + 50$, $10 \times 50 + 500$, $100 \times 5 + 500$, and $100 \times 50 + 5000$. For the three attacks marked with an asterisk, the four data sizes were instead $10 \times 5 + 5$, $10 \times 50 + 50$, $100 \times 5 + 5$, and $100 \times 50 + 500$.

Attack	$10 \times 5 + 50$	$10 \times 50 + 500$	$100 \times 5 + 500$	$100 \times 50 + 5000$
	TPR			
Sh-Timing [SW06]	0.22 ± 0.04	0.44 ± 0.02	0.10 ± 0.01	0.205 ± 0.006
Bi-Intertiming [BLJL06]	0.54 ± 0.05	0.72 ± 0.02	0.28 ± 0.01	0.398 ± 0.007
Li-Jac [LL06]	0.10 ± 0.03	0.10 ± 0.01	0.010 ± 0.003	0.010 ± 0.001
Li-NBayes [LL06]	0.84 ± 0.04	0.86 ± 0.02	0.29 ± 0.01	0.468 ± 0.007
He-MNBayes [HWF09]	0.14 ± 0.03	0.13 ± 0.01	0.024 ± 0.005	0.018 ± 0.002
Lu-Lev* [LCC10]	0.86 ± 0.05	0.91 ± 0.01	0.62 ± 0.02	0.842 ± 0.005
Pa-FeaturesSVM [PNZE11]	0.68 ± 0.05	0.82 ± 0.02	0.34 ± 0.01	0.629 ± 0.007
Dy-VNG [DCRS12]	0.43 ± 0.05	0.33 ± 0.02	0.11 ± 0.01	0.084 ± 0.004
Ca-OSAD* [CZJJ12]	0.84 ± 0.05	0.96 ± 0.01	0.73 ± 0.02	0.834 ± 0.005
Wa-OSAD* [WG13b]	0.82 ± 0.05	0.97 ± 0.01	0.78 ± 0.02	0.855 ± 0.005
Wa-FLev [WG13b]	0.00 ± 0.01	0.11 ± 0.01	0.000 ± 0.001	0.029 ± 0.002
Wa-kNN [WCN+14]	0.64 ± 0.07	0.76 ± 0.02	0.5 ± 0.02	0.704 ± 0.006
	TNR			
Sh-Timing [SW06]	0.68 ± 0.07	0.63 ± 0.02	0.57 ± 0.02	0.574 ± 0.007
Bi-Intertiming [BLJL06]	0.14 ± 0.05	0.11 ± 0.01	0.004 ± 0.003	0.0000 ± 0.0001
Li-Jac [LL06]	0.00 ± 0.01	0.000 ± 0.001	0.000 ± 0.001	0.0000 ± 0.0001
Li-NBayes [LL06]	0.04 ± 0.03	0.07 ± 0.01	0.000 ± 0.001	0.0002 ± 0.0002
He-MNBayes [HWF09]	0.04 ± 0.03	0.08 ± 0.01	0.012 ± 0.005	0.005 ± 0.001
Lu-Lev* [LCC10]	0.86 ± 0.05	0.36 ± 0.07	0.000 ± 0.001	0.0000 ± 0.0001
Pa-FeaturesSVM [PNZE11]	0.76 ± 0.06	0.85 ± 0.02	0.67 ± 0.02	0.827 ± 0.005
Dy-VNG [DCRS12]	0.02 ± 0.02	0.010 ± 0.004	0.006 ± 0.003	0.0016 ± 0.0006
Ca-OSAD* [CZJJ12]	0.00 ± 0.01	0.52 ± 0.07	0.8 ± 0.2	0.10 ± 0.04
Wa-OSAD* [WG13b]	0.6 ± 0.2	0.72 ± 0.06	0.8 ± 0.2	0.34 ± 0.07
Wa-FLev [WG13b]	1.00 ± 0.01	0.998 ± 0.002	1.000 ± 0.001	1.0000 ± 0.0001
Wa-kNN [WCN+14]	0.98 ± 0.02	0.956 ± 0.009	0.974 ± 0.007	0.975 ± 0.002

3.4.2 Time and Space Complexity

We experimentally investigate the training time, testing time, and space complexity of each WF attack. We want to know the time and space complexity in terms of the training and testing set sizes. These complexities determine if it is possible for the attack to scale towards greater training and testing set sizes, either to achieve greater accuracy or to monitor more web pages. An attacker may prefer an attack that scales better for larger sets, even if it may be less accurate.

Let n_{train} and n_{test} be the total size of the training and testing sets respectively, and let c be the number of monitored classes. c is necessarily smaller than n_{train} and n_{test}. Variables that do not change with the above variables, such as the MTU, the number of features, and the number of packets in each packet sequence are simplified as $O(1)$. For example, while each algorithm would be more expensive if each packet sequence contained more packets, the attacker and client have no control over sequence lengths and therefore we do not study this effect. For storage, we do not count the training set itself; rather, we calculate the additional storage the algorithm needs.

We give the results in Table 3.8. We do not show proof for all complexities in this table; many of them are trivial to prove and the proofs would be highly repetitive. We discuss several interesting aspects of Table 3.8 as follows.

1. In general, we observe a growing trend of increasing time and storage complexity. The algorithms that use SVMs with edit distance kernels have the highest time and storage complexity: these include `Lu-Lev`, `Ca-OSAD` and `Wa-OSAD`. This trend is reversed with `Wa-kNN`, which we designed specifically so that it would have low time and space complexity [WCN+14].

2. Normally, SVM training requires matrix inversion, which has a time complexity of $O(n_{train}^3)$. Sparse matrix inversion, however, is faster: if there are only m non-zero elements in each row, we can write the time complexity as $O(mn_{train}^2)$. LIBSVM uses a matrix inversion scheme with a yet-unknown time complexity [CL11]. LIBSVM claims that its scheme is more efficient than known theoretically optimal schemes on a variety of data sets, so we write its time complexity as $O(n_{train}^2)$.

3. For SVM testing, we write the testing time as $O(c^2 \cdot n_{test})$. This is because LIBSVM performs one-to-one testing for multi-class SVMs: For each pair of classes, SVM learns a linear separator between them, treating it as a binary classification problem, and attempts to classify the testing element against each pair; the element is then classified to the class that wins the most number of binary classifications. An alternative is one-to-all testing, where SVM learns a linear separator for each class, between its elements and all other

Table 3.8: Expected time and space complexity of the WF attacks we have surveyed, based on n_{test} and n_{train}, the total number of training and testing elements respectively, and c, the number of monitored classes.

Attack	Storage	Training	Testing
Sh-Timing [SW06]	$O(1)$	$O(n_{train})$	$O(n_{test} \cdot n_{train})$
Bi-Intertiming [BLJL06]	$O(1)$	$O(n_{train})$	$O(n_{test} \cdot n_{train})$
Li-Jac [LL06]	$O(1)$	$O(n_{train})$	$O(n_{test} \cdot n_{train})$
Li-NBayes [LL06]	$O(c)$	$O(n_{train})$	$O(c \cdot n_{test})$
He-MNBayes [HWF09]	$O(c)$	$O(n_{train})$	$O(c \cdot n_{test})$
Lu-Lev [LCC10]	$O(1)$	$O(n_{train})$	$O(n_{test} \cdot n_{train})$
Pa-FeaturesSVM [PNZE11]	$O({n_{train}}^2)$	$O({n_{train}}^2)$	$O(c^2 \cdot n_{test})$
Dy-VNG [DCRS12]	$O(c)$	$O(n_{train})$	$O(c \cdot n_{test})$
Ca-OSAD [CZJJ12]	$O({n_{train}}^2)$	$O({n_{train}}^2)$	$O(c^2 \cdot n_{test})$
Wa-OSAD [WG13b]	$O({n_{train}}^2)$	$O({n_{train}}^2)$	$O(c^2 \cdot n_{test})$
Wa-FLev [WG13b]	$O({n_{train}}^2)$	$O({n_{train}}^2)$	$O(c^2 \cdot n_{test})$
Wa-kNN [WCN+14]	$O(1)$	$O(n_{train})$	$O(n_{test} \cdot n_{train})$

elements. To classify an element, each element is scored against the classifier of each class. LIBSVM uses one-to-one testing because it has a lower training time: a total of c^2 SVMs of size n_{train}/c must be learned, giving a time complexity of $O(n_{train}^2)$, whereas one-to-all testing requires SVM to learn c SVMs of size n_{train} for a time complexity of $O(cn_{train}^2)$. In addition, previous work has found that one-to-one testing often performs better than comparable methods, such as one-to-all testing [HL02].

As we are analyzing time complexity, Table 3.8 describes how training/testing time scales for each attack, but not the actual training/testing time. For example, `Lu-Lev` is very expensive to train on our data set sizes, while `Wa-FLev` is not, though `Lu-Lev` scales better. In general, the edit distances are significantly more expensive to compute; `Wa-FLev` only approximates an edit distance, so it is much faster. We attempt to demonstrate this phenomenon next.

From our closed-world Tor data set of 10,000 elements, we ran our implementation of each attack against random data sets of 500 elements (10x50), and we recorded the total training and testing times of each attack. The standard deviation represents the variance due to randomly chosen data sets. We present the results in Table 3.9. Training and testing times are summed over all 500 elements; since all testing algorithms scale linearly with n_{test}, we can obtain the testing time of 1 element by dividing the entry in the table by 500. (This cannot be done for the training time.) The reader can derive training and testing time for any given data size by combining this table with Table 3.8. We do not claim that we implemented these attacks optimally for

Table 3.9: Training and testing time of the WF attacks we have surveyed, based on training and testing 500 elements (10×50).

Attack	Training	Testing
Sh-Timing [SW06]	1.20 s ± 0.04 s	2.11 s ± 0.05 s
Bi-Intertiming [BLJL06]	12.5 s ± 0.4 s	19.3 s ± 0.5 s
Li-Jac [LL06]	0.30 s ± 0.01 s	0.041 s ± 0.002 s
Li-NBayes [LL06]	1.50 s ± 0.04 s	0.61 s ± 0.02 s
He-MNBayes [HWF09]	1.89 s ± 0.06 s	0.199 s ± 0.006 s
Lu-Lev [LCC10]	170 s ± 20 s	38 s ± 4 s
Pa-FeaturesSVM [PNZE11]	25.5 s ± 0.7 s	2.1 s ± 0.1 s
Dy-VNG [DCRS12]	1.98 s ± 0.07 s	23.7 s ± 0.7 s
Ca-OSAD [CZJJ12]	4800 s ± 400 s	1060 s ± 90 s
Wa-OSAD [WG13b]	7200 s ± 600 s	1600 s ± 100 s
Wa-FLev [WG13b]	4.1 s ± 0.2 s	1.04 s ± 0.04 s
Wa-kNN [WCN+14]	1.76 s ± 0.05 s	2.00 s ± 0.06 s

minimal training and testing time: one may consider the results to be upper limits to the practical training/testing time of each algorithm.

We make the following observations from Table 3.9:

- The results indicate that real-time classification against a single user is possible for all of the attacks except `Lu-Lev`, `Ca-OSAD`, and `Wa-OSAD`, for which we cannot perform classification in real time for a reasonably sized training set. Levenshtein distance computation is very expensive because it requires computing a matrix of size $|P| \cdot |P'|$ for input sequences P and P'. Note that these attacks scale to the square of the data set size, so with a data set of 20,000 elements, the training/testing time would be 1,600 times larger for all three attacks. This amounts to several thousand CPU hours worth of time. We implemented these attacks in C++ making every effort to minimize the training and testing time; however, they are still several orders of magnitude more expensive than any other attack.

- It is also interesting to note that the above edit distance attacks show a much higher coefficient of variance (the standard deviation divided by the mean). We find that this is because only these attacks scale to the square of sequence length, whereas other attacks scale linearly with sequence length. If the random subset contains very large web pages, their training and testing time increases dramatically. Compare this to feature-based attacks like

Wa-kNN, which are not based on sequence length, though extracting those features may be.

- Wa-FLev does indeed meet its promise of using a significantly faster edit distance. However, Wa-FLev has a greater training plus testing time than the later Wa-kNN, which also has a much higher classification accuracy as shown previously.

- Dy-VNG has an unexpectedly high testing time. Dy-VNG uses a Naïve Bayes classifier much like Li-NBayes: to classify, it derives the probability that the packet sequence belongs to a class given its total length, time, and burst lengths. In our implementation, each burst length constitutes a separate call to a kernel density estimator. There are frequently hundreds of bursts, adding up to a significant testing time. We investigated and found out that almost all of the testing time comes from the burst lengths used in Dy-VNG. Each burst length constitutes a separate call to the KDE, even through training the KDE did not take much time.

We performed experiments on most attacks using our own desktop with a six-core processor at 1.4 GHz. We could not have run Lu-Lev, Ca-OSAD, and Wa-OSAD, however, as it would take months to classify the 100×100 data set. To deal with their expensive training and testing time, we ran these classifiers on SHARCNET, a Canadian academic consortium that offers high-performance parallel computing. Specifically, we used the kraken cluster of SHARCNET, which consists of nodes offering 4 cores with 2.2 to 2.4 GHz each. We used up to 256 cores in parallel. The computational difficulty of these attacks comes from calculating the edit distance between pairs of packet sequences. Fortunately, this process is embarrassingly parallelizable: distances between different pairs can be computed simultaneously across different cores.

3.4.3 Wa-OSAD Evaluation

We evaluate Wa-OSAD, our modification of Ca-OSAD, here. In Section 3.1.11 we described three changes to the edit distance that improved website fingerprinting classification:

1. Disabling substitutions,
2. Increasing outgoing packet operation cost, and
3. Increasing transposition cost near the start of the sequence.

Here, we evaluate each one of them on the 100×10 Tor data set. We use this data set so that `Wa-OSAD` would run in one-hundredth of the time required to attack the 100×100 Tor data set. Otherwise, we do not have enough processing power to perform the experiments in this section, even with SHARCNET. In this setting on our 100×10 closed-world data set, the base accuracy (TPR) with no modifications (that of `Ca-OSAD`) is 0.86 ± 0.01. The TPR of `Wa-OSAD` is 0.915 ± 0.009.

Disabling substitutions. One of the earliest uses for Levenshtein distance was to correct typos [OTK76], for which the possible operations in the Levenshtein distance are intuitive: insertions correspond to hitting an extra key, deletions correspond to failing to hit a key, and substitutions correspond to hitting an incorrect key. For website fingerprinting, insertions and deletions correspond to extra and removed packets due to varying content. OSAD also allows transpositions, which accounts for (adjacent) re-ordering due to random network conditions. However, substitutions do not correspond to any intuitive variance in web pages.

We disable substitutions and test the effect on `Wa-OSAD`. We observed an accuracy of 0.89 ± 0.01 on the same data set, which indicates a statistically significant increase in accuracy.

Increasing outgoing packet operation cost. There are fewer outgoing packets than incoming packets for web page packet sequences: incoming packets occur about ten times as often as outgoing packets. In addition, outgoing packets are less subject to change. For example, a news site that has four different images in its front page every day would return different amounts of content (depending on the sizes of the images), but the client would always send four requests for those images.

The default cost in `Ca-OSAD` is 2. We vary outgoing insertion/deletion cost from 2 to 12 and test the effect of the cost on classification accuracy. We found that there was almost no change in accuracy if we did not disable substitutions: the mean TPR was 0.858 ± 0.002 across the costs we tested. It is also interesting to note that the Levenshtein distance matrix when the outgoing packet insertion/deletion cost exceeds 4 is always the same. This is because the algorithm resorts to inserting an incoming packet and then substituting to insert an outgoing packet.

Next, we disabled substitutions, and ran the same experiment again, varying outgoing insertion/deletion cost from 2 to 12. We plot the results in Figure 3.1. While the small data set size (100×10) leads to a large standard deviation, we see that there is a statistically significant increase in TPR when we increase the cost from 2 to 7, after which there is a steady decrease. Indeed, when the cost is 6, we observe an accuracy of 0.916 ± 0.009, matching the overall `Wa-OSAD` accuracy for this data set. We could improve `Wa-OSAD` (very slightly) by choosing the cost as 7 instead, but we used 6 as this was the value in the original published work.

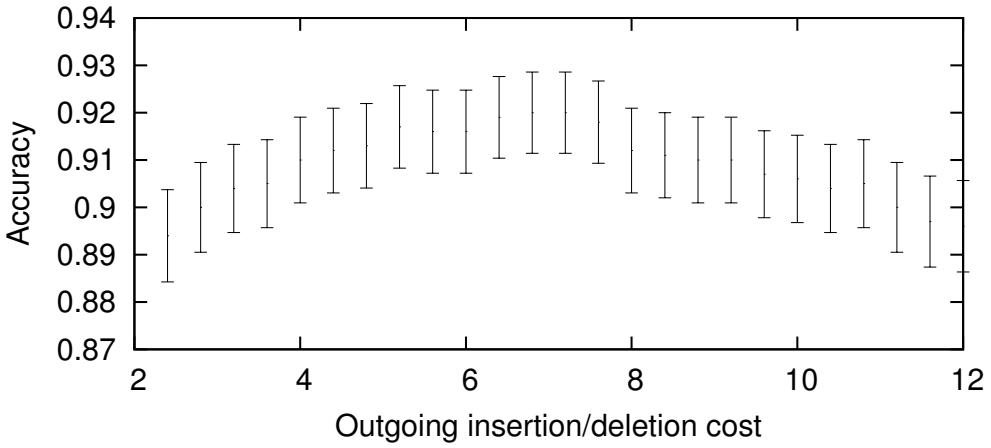

Figure 3.1: `Wa-OSAD` TPR on the 100×10 Tor data set while varying the OSAD outgoing insertion/deletion cost with substitutions disabled. Note that the outgoing insertion/deletion cost (x-axis) starts at 2; the incoming insertion/deletion cost is fixed at 2.

Increasing transposition cost at the start. Transpositions occur over the packet sequence due to random network conditions; on Tor, circuits last for 10 minutes and randomly selected volunteer nodes around the world do not offer consistent bandwidth and latency. We hypothesize that such an effect is less significant near the start of the sequence, and so we increase transposition cost near the start.

Recall that to compute $d(P^1, P^2)$ between packet sequences P^1 and P^2 we need to compute the distance between all prefixes of P^1 and P^2. For $P^1_{[:i]}$ and $P^2_{[:j]}$ we write the distance between them as $d_{i,j}$. We change the distance so that the formula for transposition cost at $d_{i,j}$ is:

$$(1 - 0.9 \min(i/|P^1|, j/|P^2|))^v$$

The transposition cost is parameterized by the power v. At the start of either packet sequence, the base is 0.1, and at the end of both packet sequences, the base is 1. The original transposition cost of `Ca-OSAD` is 0.1.

We observed no significant change in accuracy while varying transposition costs this way either with or without substitutions disabled. Given the results above, this is not surprising: increasing outgoing packet operation cost had already accounted for the increase in accuracy of `Wa-OSAD` compared to `Ca-OSAD`.

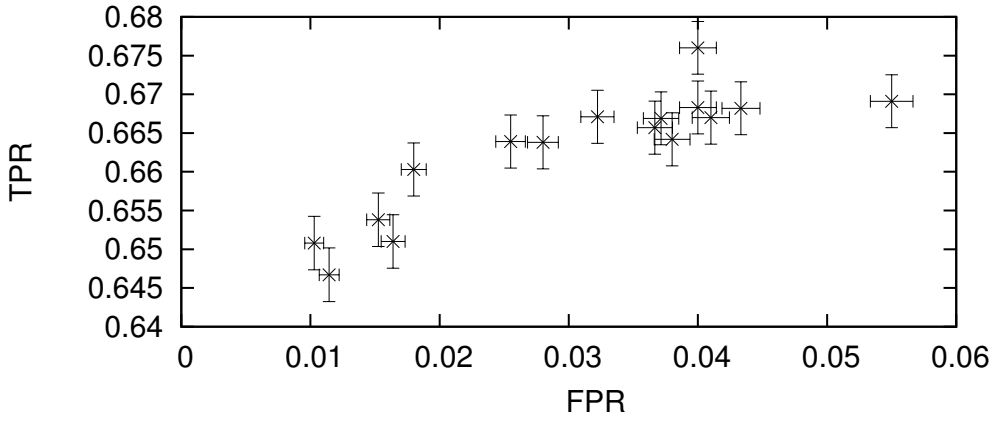

Figure 3.2: Results for TPR vs. FPR while varying $|C_0|$, the number of non-monitored pages, from 50 to 9000. The rightmost point is $|C_0| = 50$, and increasing $|C_0|$ decreases TPR, so the leftmost point is $|C_0| = 9000$. We set $k = 4$ for all data points in this plot. Note that the y-axis (TPR) does not start at 0.

3.4.4 Wa-kNN Evaluation

3.4.4.1 Parameter selection

In Section 3.1.13 we described two parameters of `Wa-kNN`: k, the number of neighbours during classification, and $|C_0|$, the number of non-monitored pages in the attacker's non-monitored training set. These parameters can trade off TPR and FPR. We emphasize that the attacker indeed has control over $|C_0|$, since it is the size of the attacker's training set, not the set of pages visited by the client; the attacker does not know or train on any non-monitored page visited by the client. As explained in Section 2.2.2, including more or fewer pages in the non-monitored data set does not affect the quality of our analysis of the open-world scenario. The attacker's decision to include more or fewer non-monitored pages does not affect the client's base rate of visiting monitored pages.

We can vary the size of the non-monitored training page to trade off TPR and FPR. We fix the number of neighbours $k = 4$, vary the number of non-monitored pages $|C_0|$ from 50 to 9000, and show the results in Figure 3.2. Increasing $|C_0|$ decreases TPR and FPR because the neighbour set of a testing point would be more likely to include non-monitored pages. We see that the effect of increasing $|C_0|$ on TPR is very minor, especially compared to the FPR. At $|C_0| = 200$, we have $a_{TPR} = 0.669 \pm 0.005$ and $a_{FPR} = 0.06 \pm 0.02$; at $|C_0| = 7000$, we have $a_{TPR} = 0.651 \pm 0.005$ and $a_{FPR} = 0.010 \pm 0.001$. As percentages, the decrease in a_{TPR} is about 2.8% and the decrease in a_{FPR} is about 81%. It is therefore almost always better for the

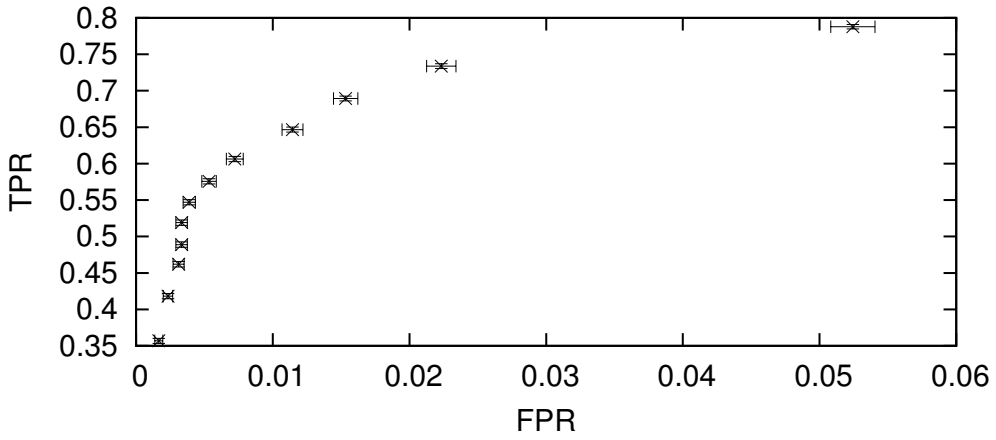

Figure 3.3: Results for TPR vs. FPR while varying k, the number of neighbours, from 1 to 15. The rightmost point is $k = 1$, and increasing k decreases TPR and FPR, so the leftmost point is $k = 15$. We set $|C_0| = 9000$ for all data points in this plot. Note that the y-axis (TPR) does not start at 0.

attacker to increase the non-monitored page set size.

We can also vary k, the number of neighbours, for a tradeoff between TPR and FPR. We fix the number of non-monitored pages $|C_0| = 9000$ and vary k from 1 to 15, showing the results in Figure 3.3. Increasing k also decreases TPR and FPR, because all neighbours must come from the same class for the classifier to return a positive classification, and the more neighbours there are the less likely this would be the case. The effect of increasing k on TPR is much more dramatic than that of increasing $|C_0|$ (compare the y-axes in Figure 3.2 and Figure 3.3). A large k allows us to reach very low values of FPR, with $k = 6$ giving $a_{TPR} = 0.576 \pm 0.005$ and $a_{FPR} = 0.0053 \pm 0.0008$, and $k = 10$ giving $a_{TPR} = 0.462 \pm 0.005$ and $a_{FPR} = 0.0031 \pm 0.0006$. Whether or not we want such a low TPR depends on the attack scenario. From another perspective, we cannot expect that an attacker would be able to capture every single access (high TPR) to a rare page (low base rate demanding low FPR).

We plot all our data for varying both k and $|C_0|$ in Figure 3.4. The larger points are the Pareto-optimal points (in TPR and FPR), and the line represents $|C_0| = 9000$, our maximum value, which does indeed match most of the Pareto-optimal points.

The optimal choice of k and $|C_0|$ depend on the expected base rate b of the monitored activity as well as the application intended by the attacker. We define the mean base rate b as the mean chance of visiting one page in the data set, the mean taken over all pages in the data set. Consider the *precision* of the attack:

$$a_{PRE} = \frac{TP}{TP + FP}$$

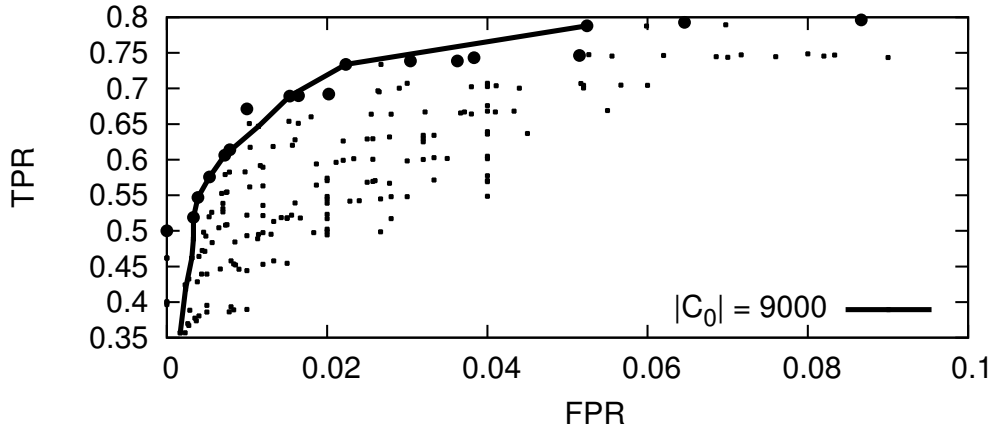

Figure 3.4: TPR vs. FPR for all combinations of k and $|C_0|$ we tested, with k from 1 to 15 and $|C_0|$ from 50 to 9000. The larger dots are Pareto-optimal points (in TPR and FPR) and the line represents $|C_0| = 9000$, which is generally Pareto-optimal. Note that the y-axis does not start at 0.

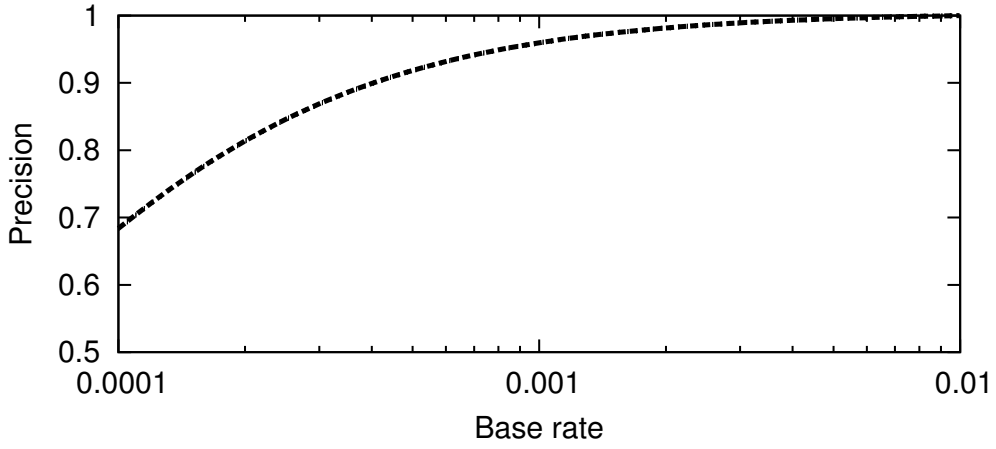

Figure 3.5: Precision of `Wa-kNN` depending on the base rate of visiting each monitored page. Note that the base rate is log-scaled, and both axes do not start at 0.

A high precision means we have avoided the base rate fallacy. Let us consider an attacker who selects parameters that maximize a_{PRE}. We show the maximal a_{PRE} that `Wa-kNN` can achieve, versus the mean base rate b, in Figure 3.5. Even if each monitored page, on average, only occupies 0.001 (one-thousandth) of the client's overall page visits, we can achieve a precision of 96%; if the mean base rate is 0.0001, then we can achieve a precision of 68%. From the data in Table 3.7, the second-best attack in the open-world scenario is `Pa-FeaturesSVM`; performing the same calculations, its precision is 29% in the first case and 4% in the latter case. Overall, `Wa-kNN` is far better than any other attack in the open-world scenario.

3.4.4.2 WLLCC

Here we show some empirical results on the properties of WLLCC, which powers `Wa-kNN`. We use the standard data set (described in Section 2.4). We choose $k = 4$ and $|C_0| = 9000$ as `Wa-kNN` parameters.

First, we investigate the parameterization of WLLCC. WLLCC has two parameters, k_{reco} (the number of points that recommend weight changes in each step) and R (the number of rounds). We want to know how those parameters affect the performance of WLLCC. Specifically, we tested the performance of WLLCC when used by `Wa-kNN` on website fingerprinting. If the wrong values have a significant negative effect, this would degrade the usability of WLLCC, as it would have to be coupled with good parameter selection.

We found that k_{reco} had almost no effect on the TPR and FPR. Varying k_{reco} from 1 to 20, the resulting `Wa-kNN` TPR was 0.794 ± 0.004 and the FPR was 0.0089 ± 0.0008. There was no clear trend of increasing or decreasing TPR/FPR; the change was so small that it could have been caused by the random re-initialization of weights. This is good for WLLCC, as it shows that WLLCC does not need to optimize over k_{reco}. For all of our experiments we simply set $k_{reco} = 5$.

For R, we show in Figure 3.6 how TPR changes with R with $|C_0| = 9000$ and $k = 4$. We observe that TPR approaches the maximum at 75 rounds, and we observed no decrease in accuracy up to 10,000 rounds. In our experiments, we found that 75 rounds takes around 0.64 s $\pm$ 0.02 s. This shows us that `Wa-kNN` requires minimal training; we saw in Section 3.4.2 that it is significantly cheaper than edit distance attacks like `Ca-OSAD` and `Wa-OSAD`, and it is practical for attacking a large number of users online.

Second, we want to empirically test the presence of local minima for our specific website fingerprinting problem. We select parameters $|C_0| = 9000$ and $k = 4$ for `Wa-kNN`, and $R = 500$ and $k_{reco} = 5$ for WLLCC. We randomly re-initialize the weight w by choosing each w_i uniformly randomly between 0.5 and 1.5, and we run `Wa-kNN` 100 times with these random

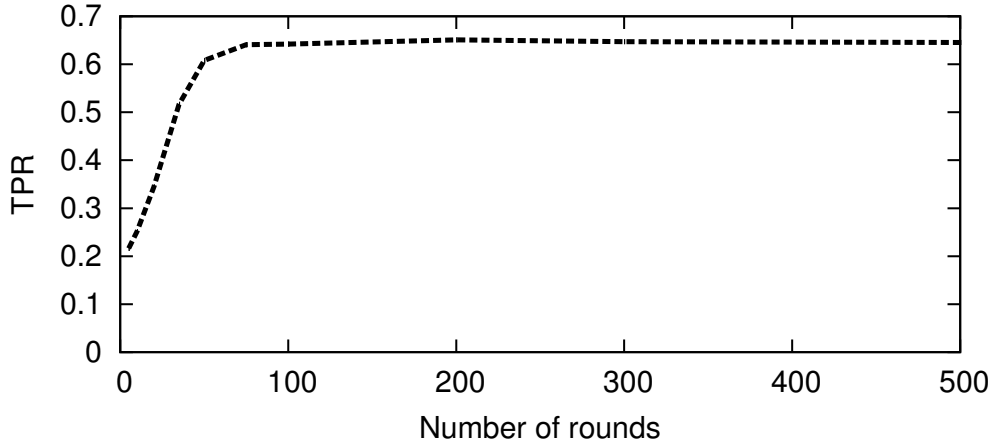

Figure 3.6: Accuracy of `Wa-kNN` depending on R, the number of rounds we run WLLCC.

weights (after WLLCC). We observe a TPR of 0.648 ± 0.002 and an FPR of 0.0076 ± 0.0006, taking the standard deviation over 100 runs. The standard deviation is very small: in fact, it it smaller than the standard deviation we derive from the binomial distribution (0.004 and 0.0008 respectively for TPR and FPR). This suggests to us that randomly re-initializing weights has little effect on WLLCC's ability to optimize weights.

3.5 Conclusions and Future Work

Website fingerprinting exposes the difference between security and privacy. We never attempt to break the encryption of the client's communication channel, but the client's page access is compromised nonetheless. Since different clients' communication are largely similar (especially on Tor Browser) when accessing the same page, we have gained significant knowledge of their browsing activities. Website fingerprinting is conceptually analogous to a dictionary attack against an encryption scheme with a small set of possible plaintexts. Without randomly padding the plaintexts (analogous to website fingerprinting defenses), even an otherwise correctly implemented encryption scheme can be broken by precomputing ciphertexts (analogous to the attacker's training set).

In this chapter, we describe several new attacks for website fingerprinting:

1. `Wa-OSAD` increases the accuracy of classification by modifying Cai et al.'s `Ca-OSAD`. `Wa-OSAD` performs well against web-browsing clients whether or not they use Tor. Our

modifications to the edit distance computation exploit our knowledge of how web pages work. Our experiments indicate that `Wa-OSAD` remains the state-of-the-art attack under the closed-world scenario. However, its training and testing time is very high, adding up to thousands of CPU hours for a data set with 10,000 elements.

2. `Wa-FLev` decreases the training and testing time of `Wa-OSAD` significantly, down to minutes under the same data set.

3. `Wa-kNN` achieves the best of both worlds: it has a very low training and testing time, but it is also nearly as accurate as `Wa-OSAD` in the closed world. It trumps all other attacks in the open world, as it has tunable parameters that trade off the TPR for the FPR and vice-versa effectively, allowing us to tackle the base rate fallacy. We have seen that our new WLLCC algorithm for `Wa-kNN` has lifted the competitiveness of the simple and fast k-NN classifier to that of SVMs for website fingerprinting. The algorithm may be useful for k-NN classification in general, and it can also be used in clustering.

We include significant new work in this thesis beyond the scope of our published papers on the above attacks. We implement 12 website fingerprinting attacks with standardized input and output to allow comparative experiments at a greater scale than previously seen work. We show new results on training and testing time. Our systematization of attacks by comparing their use of distances and features allows the possible development of hybrid attacks that combine approaches from different attacks.

We developed our attacks and performed our experiments with a strong focus on Tor, because it is one of the easiest and most popular tools for clients to achieve the level of privacy required to render website fingerprinting relevant: an encrypted channel across proxies. Tor is actively developing defenses against website fingerprinting attacks, because if website fingerprinting were perfectly successful, it would reduce the privacy of Tor to simply that of an encrypted channel. Any entry node on Tor may perform website fingerprinting to discover its clients' behaviour.

Nevertheless, website fingerprinting attacks have limitations. Some of these limitations are inherent to the problem statement, and overcoming them may be truly difficult. We categorize these limitations, which may make for interesting future work:

1. Identifying very rare pages. The low base rate of a very rare page may cause the number of false positives to overwhelm the number of true positives, and the base rate fallacy is much harder to overcome. Our experiments suggest that we can achieve high precision down to a base rate of around 0.01% (1 in 10,000), but rare pages would require more advanced techniques, and there is no limit to how rare a web page might be.

2. Identifying activity. It may be possible to identify whether a client is web browsing, chatting, listening to music, watching a video, and so on. All of these activities can be done through a regular browser, and can therefore be done on Tor. Currently, we do not tackle any activity except web browsing. Identifying them would give the attacker more information to work with.

3. Identifying web sites (as opposed to individual web pages as in this work). There is some previous work on identifying web sites [CZJJ12]. However, past results are generally limited to identifying only one or two specific web sites. It is not clear if techniques used for identifying these specific web sites can be extended to identifying hundreds of web sites at once.

4. Understanding the underlying web page. Web pages are essentially request-response pairs, and some responses trigger further requests. Currently, we collect web pages only as packet sequences, ignoring the intermediate request-response structure. If we were able to simulate packet sequences accurately from the request-response structure, with the network conditions as input, we could significantly improve website fingerprinting. Data collection would be much faster, as we would not have to wait for Tor's latency to load web pages. We would also be able to build better classifiers that ignore random noise such as advertisements.

The astute reader may have noted that we collect data sets in this chapter in a simplistic manner. We start the browser, visit a web page, record the packet sequence, and exit the browser. Realistic clients can visit several web pages, sequentially or at once, and can generate noise by engaging in other activities (e.g. listening to streaming music in the background). In our next chapter, we examine these effects on website fingerprinting and develop methods that allow all attacks in this section to adjust for realistic conditions.

Chapter 4

Realistic Attacks

In Chapter 3, we presented three new attacks on website fingerprinting: `Wa-OSAD`, `Wa-FLev` and `Wa-kNN`. We saw that `Wa-kNN` was the first attack to perform well in the open-world scenario, cutting down FPR to practically threatening levels. `Wa-kNN` is also very cheap computationally, allowing a low-resource attacker to threaten the privacy of many web-browsing clients with website fingerprinting.

However, some researchers have argued that these attacks may not be truly effective in the wild [Per13, JAA+14]. Indeed, the attacks have not been demonstrated to be effective in the wild on Tor; they were proven only under laboratory conditions. Proving these attacks directly in the wild is difficult since we can only collect packet sequences from real Tor users with their consent. Recently, Juarez et al. [JAA+14] identified significant differences between attacks in the wild and attacks proven under laboratory conditions. They noted that previous works on WF attacks made six limiting assumptions:

1. Template websites: Websites can be modeled as templates.
2. Closed world: Researchers have focused on the closed-world scenario. In the closed-world scenario, we never test WF attacks against web pages outside a fixed set of monitored pages. This is not realistic because the possible number of web pages is far larger than our training set sizes.
3. Replicability: The adversary's training set and the client's testing set are collected under similar conditions. If these conditions differ, the attacker's accuracy may suffer. Specifically, a stale training set can cause WF accuracy to deteriorate.

4. Browsing behaviour: Researchers collect packet sequences sequentially, one page after the other.

5. Page load parsing: The adversary knows when pages start and end. For example (related to the above), most users may have significant time gaps between page loads.

6. No background traffic: Researchers do not include background traffic. Equivalently, researchers assume that the adversary can filter out all background traffic.

We address these assumptions in order. Assumption 1 only concerns a particular work by Cai et al. [CZJJ12]. They showed that they can use a Hidden Markov Model to classify websites containing several pages. With larger websites this model does not scale well (since it would have too many states), but they showed that if those websites are designed with templates, their attack can still succeed. Strictly speaking Assumption 1 is not an assumption at all; it is an observation Cai et al. made from experimental results.

In Chapter 3, we showed that `Wa-kNN` performs well in the open world. In doing so, we have tackled Assumption 2. Juarez et al. did not test `Wa-kNN` possibly because it was published (in August 2014) after their research started (around June 2014). In this section, we tackle Assumptions 3 to 6, as follows:

Freshness (Assumption 3, Section 4.2). We deal with one aspect of replicability: the freshness of the training set. We determine empirically that the attacker needs only a small amount of data to perform WF effectively, and therefore it is easy to keep the data fresh. Further, we propose and test a scheme that updates the training set more efficiently by scoring each element based on consistency and relevance.

Splitting (Assumptions 4 and 5, Section 4.3 and Section 4.4). We show that it is indeed possible for adversaries to know when pages start and end from full realistic packet sequences, even if the user is visiting multiple pages at once. We turn realistic packet sequences into laboratory packet sequences by *splitting*: distinguishing between different web pages that may occur sequentially or even in parallel. We demonstrate the effectiveness of time-based splitting and classification-based splitting.

Background noise (Assumption 6, Section 4.5). We analyze three types of background noise: multimedia noise, file downloading noise, and SENDME packets used for Tor flow control. We show that noise removal is a difficult problem and we cannot do so accurately, but we also show that it is very hard to generate sufficient background noise on Tor to disrupt WF due to the design of Tor Browser.

We separate this chapter from Chapter 3 because here we do not propose a new classifier to improve the classification accuracy of known WF attacks under laboratory conditions; rather, we propose a *set of methods* to augment any WF attack with tools to operate under realistic conditions. Even with the results of this chapter, we do not know the final accuracy of website fingerprinting in the wild, because it depends significantly on user behaviour for which we have limited information. Rather, each of our methods will make website fingerprinting a more realistic threat. The code for our set of methods is available for download as before; see Appendix A.

4.1 Data Collection

We use a different data set from those described in Section 2.4, since we need a new data set to investigate splitting. We collected data between September and October 2014. The monitored web sites are also from Alexa's top 100 sites. We refer to this data set as the splitting data set, and we only use it in this chapter.

As several of our experiments required loading two pages at once in the same browser session, `tcpdump` did not provide us with enough information to distinguish between the two pages. We collected direct cell logs by modifying Tor to record the stream ID and data type of each cell, and we used the direct cell logs only to obtain the ground truth for splitting experiments. Since we only used those cell logs to obtain ground truth, this does not change the fact that any local, passive attacker can perform splitting using our methods just from `tcpdump` information.

To distinguish between cells from two different pages, we recorded the time when the request for the second page was sent, and marked the first outgoing `STREAM BEGIN` cell at or after that time as the start of the second page. We marked all new streams started after that cell as belonging to the second page, and streams before that cell as belonging to the first page. This allows us to record the ground truth of which page each cell belonged to.

4.2 Training Set Maintenance

In this section we demonstrate that it is practically feasible for a low-resource attacker to maintain a fresh training set for WF attacks on Tor. In fact, the attacker only needs to gather data constantly on a single desktop-class computer. We show the following:

1. Training set size: We empirically determine that a WF attacker can perform WF with a small training set. An effective training set is small enough that it can be kept fresh simply by updating all data points in a cycle on a single desktop-class computer.

2. Training set update: We propose an algorithm to detect and drop bad data points from a training set to keep it fresh. A trivial algorithm would be to drop the oldest data points. We propose using two other metrics for determining which points to drop: consistency and relevance. We show that these methods can be used to update the training set as effectively as simply updating all data points in a cycle, but requiring much less data collection work on the part of the attacker.

4.2.1 Training Set Size

To determine the practicality of maintaining a fresh training set, we first analyze the size of such a training set. A smaller set is naturally easier to update.

The number of packet traces the attacker needs to gather for the training set is $n_{mpage} \cdot n_{minst} + n_{nmpage}$. Here, n_{mpage} is the number of monitored sites (which we set to 100), n_{minst} is the number of instances of each site, and n_{nmpage} is the number of non-monitored sites. The attacker controls n_{minst} and n_{nmpage} to improve the accuracy of classification. We are interested in knowing how large those variables must be to ensure accurate classification.

We performed experiments on `Wa-kNN`, with the number of neighbours k set to 2. We evaluate the effect of n_{minst} and n_{nmpage} on TPR and FPR. In Figure 4.1 we held n_{nmpage} constant and varied n_{minst}, and in Figure 4.2 we held n_{minst} constant and varied n_{nmpage}. We see that a higher n_{minst} improves TPR but slightly worsens the FPR and a higher n_{nmpage} improves FPR but slightly worsens the TPR. TPR reaches above 70% and FPR reaches below 3% at $n_{minst} = 34$ and $n_{nmpage} = 5500$. At this level of accuracy, for $n_{mpage} = 100$, the number of web pages the attacker must load is 6900.

If the attacker loads more web pages, the accuracy still increases, but at a slowing rate. The attacker can further trade off an increased TPR for an increased FPR by intentionally including fewer non-monitored data points (so neighbours of each point are less likely to originate from the non-monitored class) or by decreasing the number of neighbours (as a page can only be classified as a monitored page only if all of its neighbours originated from that page).

We examine the above value of 6900 to determine whether or not it is realistic. In this data set, the mean amount of time to load each page was 12 s, which would mean that the whole training set could be re-collected once per 0.9 days on a single machine. Herrmann et al. observed only a small decrease in accuracy (within a few percentage points) up to 17 days after collection for `He-MNBayes` [HWF09]; Juarez et al. also observed that a few days of staleness does not significantly decrease accuracy [JAA+14]. We therefore claim that the attacker can maintain a sufficiently fresh training set with minimal resources. If the attacker wishes to monitor more sites, the required size of the training set would increase correspondingly.

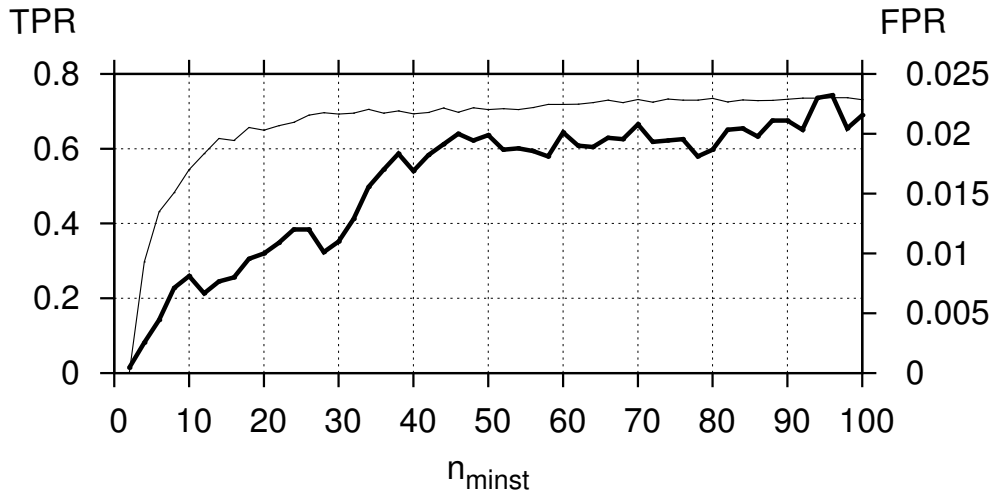

Figure 4.1: TPR and FPR when n_{minst} varies between 0 and 100, $k = 2$, and $n_{nmpage} = 9000$. The thinner line indicates TPR (left y-axis), and the thicker line indicates FPR (right y-axis).

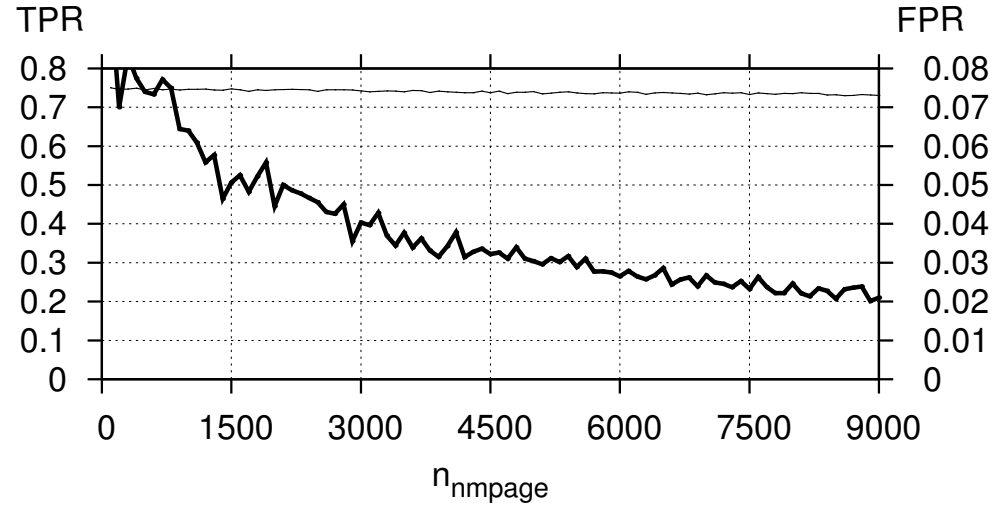

Figure 4.2: TPR and FPR when n_{nmpage} varies between 0 and 9000, $k = 2$, and $n_{minst} = 100$. The thinner line indicates TPR (left y-axis), and the thicker line indicates FPR (right y-axis).

Juarez et al. showed that if the classifier trains on one version of Tor Browser but tests on another, the accuracy may deteriorate. [JAA+14] This is because Tor has used two different WF defenses [Per13], and cell sequences appear different under different defenses. In the worst case, the attacker will have to collect three times as much data to adjust for different versions of Tor Browser, which is still practical. Furthermore, if Tor Browser is outdated, it warns the client in its start page and encourages the client to update. In any case, assuming that the client is safe because the attacker may not know about her Tor Browser version is a violation of Kerckhoffs's principle.

4.2.2 Training Set Update

We showed above that the attacker can maintain a fresh training set by collecting new data and dropping the oldest data points. However, some web pages almost never change while others change daily, so it is inefficient to simply update all data points. In this section, we design a more efficient scheme to maintain a fresh training set for distance-based website fingerprinting attacks, for example `Wa-kNN`.

We assign a score for each point and drop the lowest-scoring data points from the data set. We compute such a score by adding together a *consistency* score and a *relevance* score with random weights, as follows:

1. Consistency: The consistency score is the number of neighbours belonging to the same class as the point itself.

2. Relevance: The relevance score is the number of points from any class that have this point in its top-20 closest neighbour set. We chose 20 as a number significantly greater than the neighbour set we used for classification in k-NN (which is between 1 and 5).

We added together those two scores with random weights to show that both of these scores are indeed useful, and they can be combined in any way; we did not overfit our score.

To test if our score is effective for updating the whole training set, we constructed the testing set by including only the freshest half of the original data set, and we constructed the training set by including the highest-scoring half of the original data set. We used leave-one-out cross validation, so that while testing an element from the testing set, if it also appears in the training set, we did not allow the classifier to use that element in the training set. If classification of the testing set is accurate with such a training set, then the score is effective. We compared our score with several baselines:

Table 4.1: TPR of `Wa-kNN` after updating the training set using our scheme with relevance and consistency scoring (Re-con), and several other baseline comparisons (Stale, Random, Fresh).

Update method	Accuracy (TPR)
Stale	0.711 ± 0.001
Random	0.756 ± 0.005
Fresh	0.772 ± 0.001
Re-con	0.775 ± 0.003

1. Stale: The training set is the least fresh half of the original data set. The attacker does not update the training set.

2. Random: The training set is randomly selected. The attacker updates random points in the training set.

3. Fresh: The training set is the freshest half of the original set (as is the testing set). The attacker updates all points in the training set.

The Stale training set is older than the Fresh training set by about a week. We show the results in Table 4.1 by generating the training sets 100 times with the above methods and classifying the testing set with the generated training set. We show only the TPR as there was no significant change in the FPR. Our new scheme is listed as Re-con (relevance and consistency) in the table. The low standard deviation of the accuracy value convinces us that our updating scheme is significantly more useful for classification than simply a trivial random scheme or the Stale training set, and it is about as useful for classification as the Fresh training set (which is the most expensive to maintain). Therefore, the attacker can indeed maintain an effective training set without having to update all data points.

4.3 Splitting Algorithms

Current WF techniques only accept cell sequences corresponding to a single page as input; under laboratory conditions, the researcher uses the ground truth of the data collection system to decide where to split the full sequence of cells into single-page cell sequences. Therefore, these attacks cannot operate in the wild unless we can split accurately without this ground truth.

In this section, we tackle the splitting problem. Solving the splitting problem incorrectly could result in a cell sequence with extra or fewer cells (harder to classify); missing a split

altogether almost certainly results in two negatives (two pages classified as a non-monitored page that the attacker is not aware of). We explain the terminology used in this section, outline our strategy to solve the splitting problem, and then discuss the specific algorithms we use.

4.3.1 Terminology

In this section we introduce some new terminology in order to explain our splitting solution.

As described in Section 2.3, we collect data as Tor cell sequences. The user may visit many web pages over a long period of time (for example, an hour), and the attacker collects the sequence of incoming and outgoing cells as a *full sequence*. A full sequence is a type of cell sequence that contains many page visits.

In splitting, the attacker wants to divide the full sequence into *cell segments*. The ultimate goal is to have cell segments that each contain a single cell sequence (corresponding to one web page). These are referred to as *single-page segments*. Cell segments that contain more than one page, possibly in error or as a temporary transitional state between the full sequence and single-page segments, are referred to as *multi-page segments*.

To obtain single-page segments, the attacker needs to find the correct *splits*. A split is a location in the full sequence where the cells before the split and the cells after the split belong to different web pages. In multi-page segments where multiple pages may overlap, we define a split as the location of the first cell of the latter page. In such a case, splitting correctly would still result in some overflow of cells into the latter segment. We do not attempt to split between different streams (for example, different images or scripts) of the same web page.

4.3.2 Splitting Process

There are three steps in our splitting process, as shown in Figure 4.3:

1. **Time-based splitting**. The full sequence is split with a simple rule: if there is a time gap between two adjacent cells greater than some amount of time t_{split}, then the sequence is split there into individual cell segments.

2. **Classification-based splitting**. Time-based splitting may not be sufficient to split multi-page segments with a smaller time gap than t_{split}. We use machine learning techniques both to decide whether or not to split them further (*split decision*) and where (*split finding*).

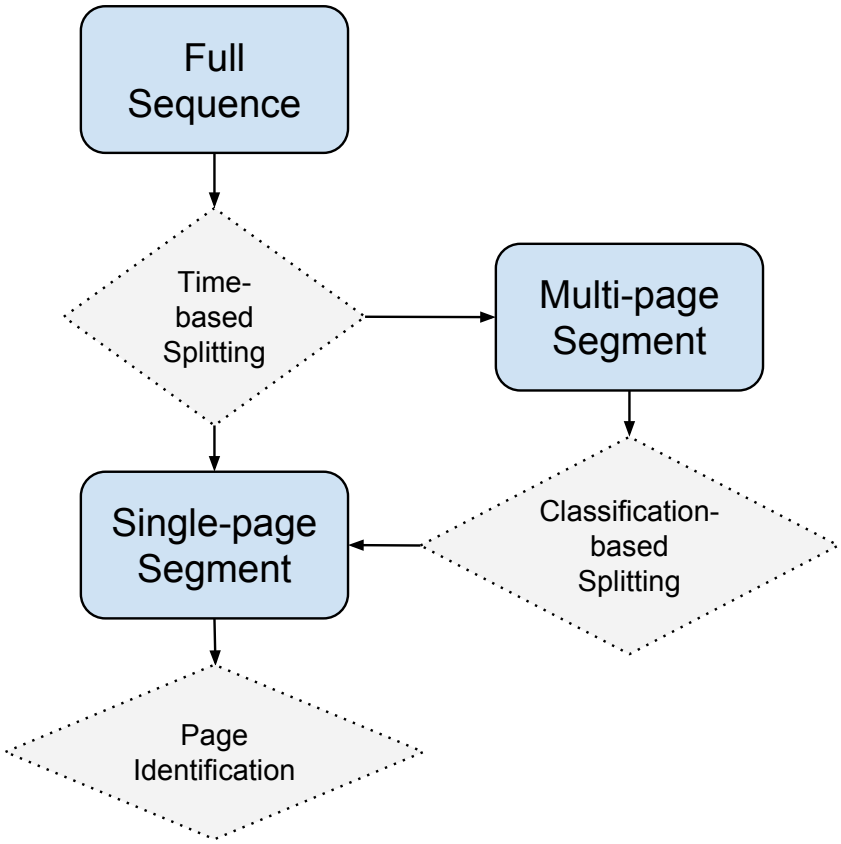

Figure 4.3: Splitting process, with ideal results. Overall, the attacker wants to obtain single-page segments from the full (raw) sequence.

3. **Page identification**. The objective of the first two steps is to split the full sequence into single-page segments as accurately as possible. Then, we may attempt to identify the page corresponding to the cell segment. In this section we will investigate the effect of background noise, incorrect splitting, and incomplete pages on known page identification techniques, which we developed and evaluated in Chapter 3.

The following types of segments may result from time-based splitting:

1. Single-page segments. In this case we should not attempt to split the segment further.

2. Two-page segments. We attempt to split these cell segments by finding the point where the second page starts.

3. Multi-page segments containing three or more pages.

We apply classification-based splitting in two steps in order to split two-page segments further into single-page segments. First, in *split decision*, we attempt to distinguish between single-page and two-page segments with machine learning. Then, in *split finding*, we take two-page segments and find the optimal location to split them. To find a split in a two-page segment, the classifier looks at each outgoing cell and assigns a score based on features of its neighbouring cells. The classifier then predicts that the maximally scored cell is the correct split.

We do not split multi-page segments containing three or more pages. Our methods may apply to these segments as well, but the accuracy would be lower, and it complicates our presentation. Rather, we show in Section 4.4.1 that they are unlikely to occur, and we minimize their occurrence probability as an explicit strategy.

4.3.3 Time-based Splitting

Cell sequences from page loading may be separated by a time gap during which there is no web activity. We therefore split the full sequence at all points in the sequence where no traffic is observed for some amount of time t_{split}. Our choice of t_{split} seeks to minimize the chance of splitting single-page segments, which should not be split any further. A larger t_{split} reduces such a risk but renders two-page and multi-page segments more likely if the client's *dwell time* is small. (The dwell time is the amount of time a user stays on a page between two page visits.) To split these we need to apply classification-based splitting, which we discuss in the next section.

To obtain the correct t_{split} we consider two potential errors resulting from splitting with t_{split}:

1. Splitting a single-page segment with t_{split}. The attacker should not split single-page segments further. We will consider the consequences of such a split: it is still possible to classify a cell sequence correctly even with only part of the cell sequence.

2. Failing to split a two-page or multi-page segment with t_{split}. The probability of this error occurring depends on dwell time. We must proceed to classification-based splitting in order to split two-page segments, which is less accurate than a simple time-based rule.

We want either source of error to be unlikely. A smaller t_{split} increases the chance of the former and decreases the chance of the latter, and vice-versa, so a suitable value must be chosen. We will show how we choose t_{split} and how it affects accuracy values in Section 4.4.1.

4.3.4 Classification-based Splitting

Cell segments used in classification-based splitting can consist of web pages organized in four possible ways:

Class 1. Two pages, positive-time separated. This is where the user dwells on a web page for an amount of time before accessing the next, thus causing a lull in web activity and a noticeable time gap in traffic. This noticeable time gap is, however, less than t_{split} used by time-based splitting; otherwise it would have been split in time-based splitting.

Class 2. Two pages, zero-time separated. This is where the user clicks on a link from a web page that is loading, thus halting the web page and sending out requests for the next immediately, so that there is a clear division between two web pages but it is not marked by a time gap.

Class 3. Two pages, negative-time separated. This is where the user is loading two pages at once in multiple tabs. In this case, we consider a correct split to be the time at which the second page starts loading. This is the hardest class to split as there is no noticeable gap nor a clear pattern of cells indicating the gap. However, we can still split cell sequences in this class using machine classification by extracting useful features. We describe the feature set we use later in this subsection.

Class 4. One page. In this case time-based splitting was sufficient to isolate a page into its own cell segment. We need to avoid splitting such a page.

To split two-page segments properly is a two-step process. The first step is *split decision*, where we distinguish between two-page segments and single-page segments. This is necessary to perform the second step, *split finding*, where we find the split in two-page segments.

Split decision. The machine for split decision takes as input a page segment, and returns a binary "yes" (it is a two-page segment) or "no" (it is a single-page segment). It is trained on two classes: a class of two-page segments, and a class of single-page segments. If split decision returns "no", we believe the sequence comes from class 4, so we skip split finding and go straight to page identification. For *split decision*, we tried three algorithms: `SDec-kNN`, `SDec-TimekNN`, and `SDec-SVM`.

Split finding. When the split decision machine returns "yes", we move on to *split finding*. The correct split location is the point at which the second page begins loading (i.e. an outgoing request cell is sent to the server of the second page). To find the correct split, we score

every outgoing cell in the cell segment based on its neighbourhood of cells, and return the highest-scoring outgoing cell as the location of the classified split. The split-finding machine takes as input a cell and its neighbourhood of cells, and returns a score representing its confidence that this cell marks the start of the second page. For *split finding*, we tried three algorithms: `SFind-kNN`, `SFind-LFkNN` and `SFind-NB`.

We describe the algorithms here.

Split Decision.

1. `SDec-kNN`: k-NN with features and weight learning. This is similar to `Wa-kNN`: we extract features from the cell segment and use WLLCC to determine the distance function for a k-NN classifier. We tested this algorithm with the same feature set as `Wa-kNN`.

2. `SDec-TimekNN`: Time-based k-NN. We added a number of features to the above that are related to interpacket timing. These include the largest and smallest interpacket times in the cell segment, the mean and standard deviation for interpacket timing, and others.

3. `SDec-SVM`: SVM with features. This is similar to the approach used by Panchenko et al. for website fingerprinting [PNZE11]. The chief difference is that we choose a different cost and gamma value (as this is a different problem); in addition, we do not append the entire cell segment onto the feature list. We selected parameters for SVM as it is highly sensitive to incorrect parameters. We chose the radial basis function kernel (see Section 3.2.3) with $\gamma = 10^{-13}$ and cost for incorrect classification $C = 10^{13}$ to maximize accuracy.

Split Finding.

1. `SFind-kNN`: A k-NN classifier with a scoring system. As the features in `Wa-kNN` are not suitable for split finding, we used a set of 23 features based on timing and packet ordering, given in Table 4.2. We score each candidate packet by finding the 15 closest neighbours: Neighbours from the "correct split" class increase the score and neighbours from the "incorrect split" class decrease the score. We guess that the highest-scoring candidate packet is the real split.

2. `SFind-LFkNN`: A k-NN classifier that uses the last cells before splits and first cells after splits to classify elements. The classifier recognizes four classes: *correct-before*, the last 25 cells before a correct split; *correct-after*, the next 25 cells after a correct split, and similarly *incorrect-before* and *incorrect-after* for incorrect splits. When evaluating a candidate split,

Table 4.2: Full feature set of SFind-kNN, with 23 features. Features are extracted from each *cell* (rather than each packet sequence) by examining its neighbourhood of cells. We write the candidate cell as P_i. As usual, if a feature is not available, we denote it with null.

Number of features	Description
5	$P_{T_{i-2}}$, $P_{T_{i-1}}$, P_{T_i}, $P_{T_{i+1}}$, $P_{T_{i+2}}$ (five intercell times around the candidate).
3	Let M be the set of all intercell times before and after the candidate cell, up to fifty cells on each side. We take the mean, standard deviation, and largest element of M.
1	$P_{T_i} - P_{T_{i-50}}$.
1	P_{t_i} (intercell time).
9	$P_{T_{i+2k}} - P_{T_{i-2k}}$ for k from 1 to 9.
4	Number of incoming cells, and also the number of outgoing cells, both five and ten cells before and after P_i.

the last 25 cells before the candidate split are scored against *correct-before* and *incorrect-before* and the next 25 cells are scored against *correct-after* and *incorrect-after*. The feature set for `SFind-LFkNN` is identical to the feature set for `SFind-kNN`, except that it only involved cells in one direction, either before or after the candidate cell.

3. `SFind-NB`: Naïve Bayes with features. The Naïve Bayes classifier involves explicit probabilistic scoring, which is suitable for this purpose, and it was used in several WF attacks. Features are trained and tested with the assumption that they follow independent normal distributions. We use the same feature set as `SFind-kNN` above. Each potential split will have a score indicating the possibility that it belongs to the first class, and the potential split with the highest score is picked out.

We intentionally chose methods similar to ones that succeeded for WF as WF is similar to splitting. We do not use edit distances like `Lu-Lev`, `Ca-OSAD` and `Wa-OSAD` because edit distances ignore timing, and timing is important to finding splits.

4.3.5 Pre-splitting

In the above, we proposed using split decision to distinguish between single-page segments and two-page segments. In our work, we also tested a process called *pre-splitting*, an extended version of split decision which further distinguishes between all three types of two-page segments

(positive-time, zero-time, negative-time). This way, split finding will be easier as the attacker can train on the same type of splits (the algorithms we use are otherwise the same). For instance, a user loading two pages in two tabs looks different from a user loading two pages in the same tab; using one type of data to find a split in the other is harder.

In our work [WG15a], we tested pre-splitting on all split decision algorithms as above, and applied split finding to the results. In our best combination of split decision and split finding algorithms, we found that there was very little difference in accuracy with or without pre-splitting. To keep the presentation of our results simple, we will not present results on pre-splitting in this thesis.

4.4 Splitting Results

In this section, we experimentally validate our splitting algorithms and show their accuracy. The main results are:

1. **Time-based splitting** (Section 4.4.1). Empirically, we find that we can perform time-based splitting with $t_{split} = 1$ s, at no cost to page identification accuracy. We will discuss previous research, which suggests that most web page accesses have a higher dwell time than 1 s.
2. **Classification-based splitting** (Section 4.4.2). If the above fails to split a two-page segment (i.e., the time gap is smaller or did not exist), we find that we can still perform *split decision* to identify two-page segments and *split finding* to split them correctly with high accuracy.

Combined, these results mean that we can split full sequences into single-page segments with high accuracy.

4.4.1 Time-based Splitting

The first step in processing the full sequence into single-page segments is to split the full sequence with a simple time-based rule: if the difference in time between two cells exceeds t_{split}, then we split the sequence there.

We argued in Section 4.3.3 that there are two opposing positive factors motivating the choice of t_{split}:

1. Increasing t_{split} decreases the chance of splitting single-page segments, which we should not split.

2. Decreasing t_{split} increases the chance of splitting two-page or multi-page segments, which we should split.

We will present our experimental results in this section accordingly. First, we will increase t_{split} until there is almost no chance of splitting single-page segments. Then, we will decrease t_{split} to increase the chance of splitting two-page or multi-page segments. It may seem that those two steps should be taken in tandem with each other to find an optimal solution; however, we will in fact find that t_{split} is so small after the first step that we do not need to take the second step at all.

Increasing t_{split}. To experimentally determine the suitable t_{split}, we applied time-based splitting to single-page segments, and tested page identification with `Wa-kNN` on the resulting segments while increasing t_{split} from 0.01 s to 1.5 s. We ran our experiments on the splitting data set, on which the accuracy of `Wa-kNN` was 0.89.

If t_{split} splits a single-page segment into several segments, we choose the largest segment to keep to classify, and assign the smaller segments to the non-monitored class. We consider two attackers, an *adjusting attacker* and a *non-adjusting attacker*:

Adjusting attacker: The attacker applies splitting to his own training set as well, under the same t_{split}, in order to maximize his accuracy in classifying split segments. This is an entirely realistic strategy, since the attacker has control over what t_{split} to use on both testing and training elements.

Non-adjusting attacker: We will also consider an attacker who does not split his training set. In other words, his training set is simply the original data set.

We do not make any adjustments to the algorithm of `Wa-kNN` itself for splitting.

In Figure 4.4 we plot the resultant negative effects on the true positive rate (the effect on the false positive rate is very small). We see from Figure 4.4 that classification accuracy reaches the maximum after around $t_{split} = 1$ s. At this value, $a_{TPR} = 0.88$ compared to the maximal 0.89. There are in fact time gaps still greater than 1 s, but removing them did not affect classification accuracy.

If the attacker does not adjust, the accuracy is significantly lower. For example, at $t_{split} = 0.5$ s, splitting with adjustment gives $a_{TPR} = 0.83$, splitting without adjustment gives $a_{TPR} = 0.68$. At $t_{split} = 1$ they were 0.88 and 0.81 respectively.

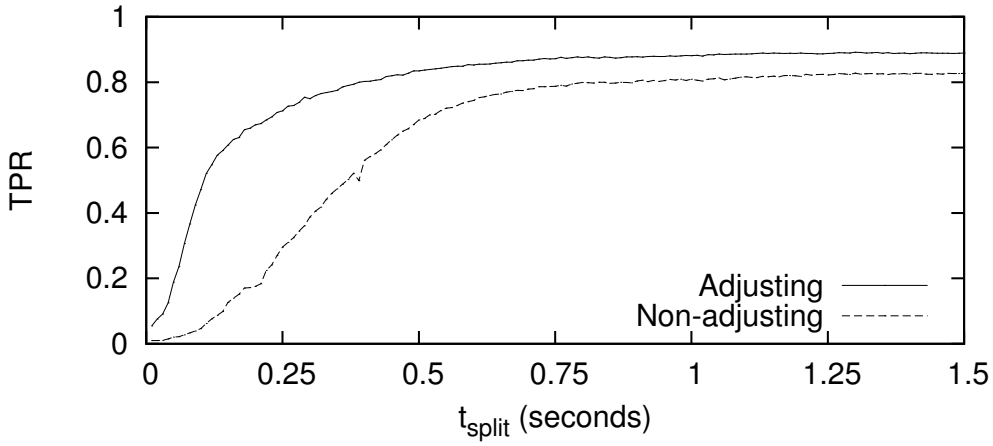

Figure 4.4: Change in TPR of `Wa-kNN` when splitting cell sequences with t_{split} for the adjusting attacker and non-adjusting attacker.

We have therefore found that there is almost no loss in classification accuracy with $t_{split} = 1$ s. The implication is that even if the client only waits 1 s between two page loads, we can distinguish between them by time-based splitting. Next, we want to know if we should reduce t_{split} any further.

Decreasing t_{split}. It may be argued we should reduce t_{split} below 1 s, to trade an increased chance of splitting single-page segments for an increased chance of splitting two-page or multi-page segments.

Reducing t_{split} to some $t'_{split} < 1$ s has a benefit if the client's dwell time t_{dwell} is between t'_{split} and t_{split}. Figure 4.4 suggests that t'_{split} can be as low as 0.1 s. Recall that classification-based splitting does not handle segments with three or more pages, if there are two or more dwell times less than t_{split} in a row. We want this situation to be rare after time-based splitting. In other words, we want to know the probability $t'_{split} < t_{dwell} < 1$ s.

As we cannot collect information on Tor clients, we do not know the true dwell time distribution of Tor clients, so we must defer to previous work on dwell time for web traffic (without Tor). It has long been established that dwell time has a heavy tail [CB97]. Previous authors have found dwell time to be well-fitted by Pareto [CC01], lognormal [MCF00] and "negative-aging" Weibull [MCF00, LWD10] distributions with the mean being around 60 s. The probability of 0 s $< t_{dwell} <$ 1 s is thus very small, so any choice of t'_{split} is not likely to produce a benefit. This also implies that the probability of failing to split a time gap twice in a row (for multi-page segments with three or more pages) is much smaller.

Furthermore, we will next see that when there exists a small time gap between packet sequences (under 1 s), denoted as Class 1 in our terminology (see Section 4.3.4), classification-based splitting is very accurate. This further contributes to our argument that reducing t_{split} further would produce no notable benefit in splitting two-page or multi-page segments.

4.4.2 Classification-based Splitting

In this section, we present our results on splitting two-page segments with classification. We used two metrics to evaluate the effectiveness of splitting:

1. Split deviance. Writing the guessed split position as s, and the true split as s_{true}, the split deviance is $s - s_{true}$. We also present the absolute split deviance. A larger deviance increases the difficulty of page identification.

2. Split accuracy. We consider the split correct if

$$|s - s_{true}| < 25$$

   We choose 25 as the range within which the page identification TPR remains above 0.7 (we show this in the next paragraphs). From our data set, we computed that randomly guessing any outgoing cell as the correct split will result in a split accuracy of 4.9%; this number serves as a baseline comparison for our methods. We also present the standard deviation of split accuracy by randomizing parts of the data set for training and testing.

We measured the effect of split deviance on page identification accuracy with `Wa-kNN`. First, we took the last ℓ cells from a single-page segment and prepended it to another single-page segment, and tested the effect of varying ℓ on the accuracy of identifying either page; this simulates the situation where the split was too early. Similarly we tested the effect of varying ℓ on accuracy when the split was too late. We tested `Wa-kNN` on the 100×40 Tor data set. We use the closed-world scenario to maximize the TPR, giving the attacker the most optimistic assessment while measuring the effect of split deviance on classification. We make no adjustment to `Wa-kNN` to tolerate incorrect split deviance.

We show the results in Figure 4.5. We see that there is almost no loss in accuracy when cells are appended to the end of a single-page segment (positive split deviance, thicker line), or when cells are removed from the end of a single-page segment (negative split deviance, thinner line). In either case, classification is still very accurate. However, accuracy drops when cells are prepended to the start of a single-page segment, or when cells are removed from the start of a

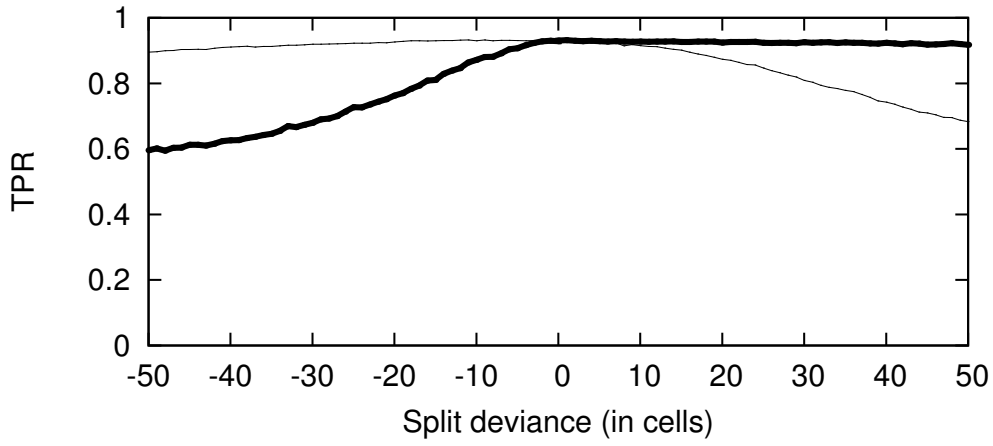

Figure 4.5: TPR of `Wa-kNN` depending on split deviance. The thicker line is for the segment that received extra cells because of split deviance; the thinner line is for the segment that lost cells.

single-page segment. The TPR drops more quickly in the former case. This is because `Wa-kNN`, drawing from established results in `Pa-FeaturesSVM` and `Wa-OSAD`, uses the initial cells as a significant feature for classification. At a split deviance of 25 cells, the accuracy is 0.92 for the first segment (gained cells at its end) and 0.85 for the second segment (lost cells from its start). at a split deviance of -25 cells, the accuracy is 0.92 for the first segment (lost cells at its end) and 0.72 for the second segment (gained cells at its start).

Data collection. We loaded zero-time separated pages (class 2) by loading two pages in the same tab, and we loaded negative-time separated pages (class 3) by loading two pages in different tabs, with a time gap between 5 and 10 seconds. We chose this time gap to give enough time for the first page to start loading, and to ensure that the chance of the first page finishes loading before the second page starts is small. If the first page finishes loading before the second page starts, then these two pages are actually separated by a positive time gap. We processed all cell sequences to find positive-time separated two-page segments this way, and moved them from their original classes to positive-time separated pages (class 1). Overall, the number of elements in classes 1, 2, 3 and 4 were about 1800, 1200, 1500, and 1600 respectively.

Split decision. We evaluate the accuracy of *split decision*: distinguishing between single-page segments and two-page segments. We collected 5,000 two-page segments and 5,000 single-page segments from the sensitive site list. We tested `SDec-kNN`, `SDec-TimekNN`, and `SDec-SVM` for this problem, and show the split accuracy and absolute split deviance.

We present the results in Table 4.3. The three classifiers had similar performance, except that `SDec-SVM` had trouble identifying class 4 (single-page segments). Since identifying class 4 as class 1/2/3 means that it will undergo splitting by the split finding process, and splitting a single-page segment makes classification significantly harder, we want class 4 to be identified correctly. Therefore the other two methods are superior; we chose `SDec-TimekNN` for further experiments.

Split finding. If *split decision* tells us that there are two pages in our segment, we need to find the location of the split to separate them. We evaluate the accuracy of *split finding* in two-page segments. We tested `SFind-kNN`, `SFind-LFkNN`, and `SFind-NB` for this problem, and similarly present the split accuracy and absolute split deviance.

We present the results in Table 4.4. For both cases, our results showed that `SFind-NB` was only slightly better than random guessing, even though it used the same features as `SFind-kNN`. This is possibly because `SFind-kNN` was more tolerant to bad features than `SFind-NB`, as the `SFind-kNN` weight-learning process filtered out bad features. `SFind-LFkNN` was overall slightly worse than `SFind-kNN` in terms of finding the correct split. This may be because `SFind-LFkNN` is only allowed to train on a maximum of 25 cells before and after true splits, so it has no access to features relating to the remainder of the cell sequence (such as total number of cells in the sequence).

Finally, we combine all the results in this section and present split accuracy using the best algorithms (`SFind-TimekNN` for segment classification and `SFind-kNN` for split finding) in Table 4.5. The table shows that classification of Class 3 is indeed the most difficult: in fact, we cannot expect to correctly identify overlapping pages.

Table 4.3: **Split decision accuracy.** The number in row i, column j is the probability that the classifier thought an element of class i belonged to class j. A greater value in the diagonal is better.

		Class 1/2/3	Class 4
SDec-kNN	Class 1/2/3	0.92 ± 0.04	0.08 ± 0.04
	Class 4	0.022 ± 0.006	0.978 ± 0.006
SDec-TimekNN	Class 1/2/3	0.96 ± 0.05	0.04 ± 0.05
	Class 4	0.03 ± 0.02	0.97 ± 0.02
SDec-SVM	Class 1/2/3	0.93 ± 0.01	0.07 ± 0.01
	Class 4	0.10 ± 0.03	0.90 ± 0.03

Table 4.4: **Split finding accuracy.** We show the accuracy for each of the three types of two-page segments (Class 1: positive-time, Class 2: zero-time, Class 3: negative-time). We show the first, second, and third quartiles of the absolute split deviance in parentheses, separated with slashes.

	Class 1	Class 2	Class 3
SFind-kNN	0.92 ± 0.02 $(0/0/1)$	0.66 ± 0.04 $(2/8/59)$	0.34 ± 0.04 $(9/87/332)$
SFind-LFkNN	0.94 ± 0.01 $(0/0/2)$	0.61 ± 0.03 $(3/13/53)$	0.18 ± 0.02 $(60/205/526)$
SFind-NB	0.16 ± 0.03 $(67/339/1507)$	0.09 ± 0.02 $(237/851/2126)$	0.04 ± 0.02 $(388/1385/3922)$

Table 4.5: Overall split accuracy for the best algorithms (`SDec-TimekNN` and `SFind-kNN`). Classes 1, 2, and 3 are positive-time, zero-time, and negative-time separated two-page segments respectively; Class 4 is single-page segments. For class 4, split accuracy is simply the chance that it was identified as class 4, as informed by split decision.

Class 1	Class 2	Class 3	Class 4
0.88 ± 0.05	0.63 ± 0.05	0.32 ± 0.04	0.97 ± 0.02

4.5 Removing Noise

In this section, we will discuss another aspect of realistic packet sequences that may deteriorate theoretical accuracy values: background noise. We discuss two types of background noise on Tor.

1. Control cells. Tor uses SENDME cells for flow control; other types of control cells are too rare to significantly affect website fingerprinting. SENDME cells interrupt bursts.
2. Content cells. The client may be loading other content while web browsing, such as downloading a file, listening to music, or watching a video.

These activities are persistent and may interfere with both splitting and page identification if the noise transmission rate is high enough. We characterize noise in Section 4.5.1. We refer to cells that are not noise as *real cells*: we want to distinguish between noise cells and real cells. Then, we present three results in this section:

1. For control cells, it is possible to remove SENDME packets on Tor with some accuracy by using timing (Section 4.5.2). We describe a procedure to remove SENDME packets, and we evaluate it by applying it before attempting to classify.

2. Removing content cells as noise is difficult (Section 4.5.3). We demonstrate the difficulty both practically and analytically. Practically, we will describe how our classification-based and counting-based approaches both failed to remove noise accurately. Analytically, we will show that there are inherent difficulties in removing this noise due to variation in intercell timing.

3. Generating noise is difficult with Tor Browser (Section 4.5.4). Due to the design of Tor Browser, it is difficult to generate sufficient noise on Tor to disrupt WF, as Flash is disabled and file downloading is slow and discouraged.

4.5.1 Characterization of Noise

We first characterize noise by identifying the noise rate (measured in cells per second) that is necessary for WF accuracy to drop. To do so, we add random incoming cells to cell sequences and ask `Wa-kNN` to identify the original page. We add noise cells at n_{noise} cells per second, with each intercell time selected uniformly randomly between 0 and $2/n_{noise}$ seconds.

We show the results in Figure 4.6, varying n_{noise} between 0.2 and 200. We use the same methodology behind Figure 4.5 to generate this graph: we apply `Wa-kNN` on the 100×40 Tor data set, with a base accuracy of 0.93 when there is no noise. We test two types of attackers: the first attacker is a non-adjusting attacker who makes no attempt to adjust for the presence of noise cells; he does not add noise to the training set. The other attacker is an adjusting attacker who recognizes both the noise and the noise rate, and adds noise to the training set using the same mechanism and noise rate.

Non-adjusting attacker. The non-adjusting attacker is significantly devastated by the presence of noise cells: to select a few values from the graph, the TPR is 0.89 at 5 cells per second, 0.80 at 10 cells per second, 0.63 at 20 cells per second, and 0.11 at 100 cells per second. As a comparison, the mean number of real cells per second when web-browsing in our data set is 83 cells per second. 20 cells per second is equivalent to 80 kbps. In general, video streaming, audio streaming and file downloading may exceed this bit rate, while other sources of noise such as chatting and AJAX may not.

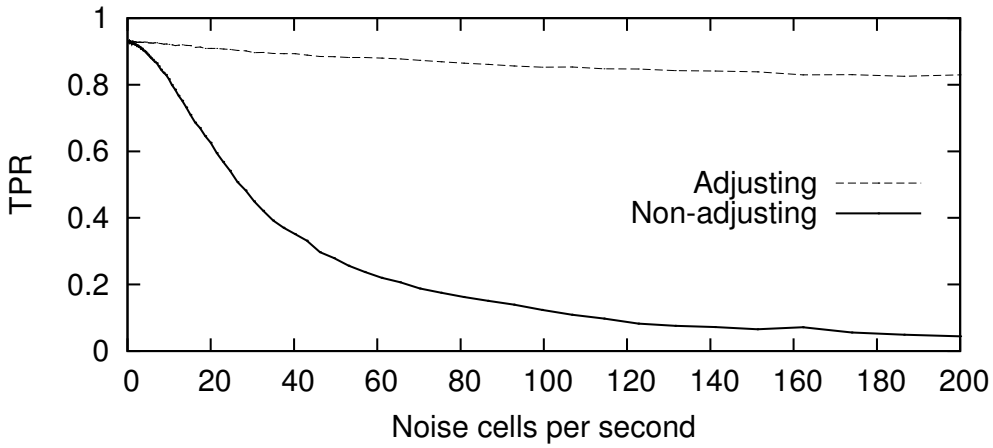

Figure 4.6: Change in TPR of `Wa-kNN` when noise cells are added randomly to cell sequences. Solid line represents a non-adjusting attacker; dotted line represents an adjusting attacker (definitions in the text).

Adjusting attacker. The adjusting attacker is only barely affected by the presence of noise cells. The TPR is 0.93 at 5 cells per second, 0.92 at 10 cells per second, 0.91 at 20 cells per second, and 0.85 at 100 cells per second. Since the intercell times are still random, the adjusting attacker cannot easily predict noise cell locations. An adjusting attacker is not trivially easy to implement, since the rate of noise changes significantly over time and there are many different types of noise (more discussion later in this section). It is possible to identify the rate and type of noise between single-page segments. While an adjusting attacker may not be realistic in all situations, it represents an application of Kerckhoffs's principle.

4.5.2 Removing SENDME Noise

As described in Section 2.1.2, Tor issues a circuit-level SENDME cell every 100 cells on each circuit and a stream-level SENDME cell every 50 cells on each stream. We attempt to automatically remove these cells as they provide us with no information, just as we remove ACK packets at the TCP level.

In this section, we attempt to remove outgoing SENDME cells. We do not attempt to remove incoming SENDME cells because they are significantly less frequent. In the context of this section, SENDME cells belong to the positive class and real cells belong to the negative class. For example, trying to remove a real cell (thinking it was a SENDME cell) is a false positive event. We test two SENDME removal algorithms, *Decrement* and *Reset*:

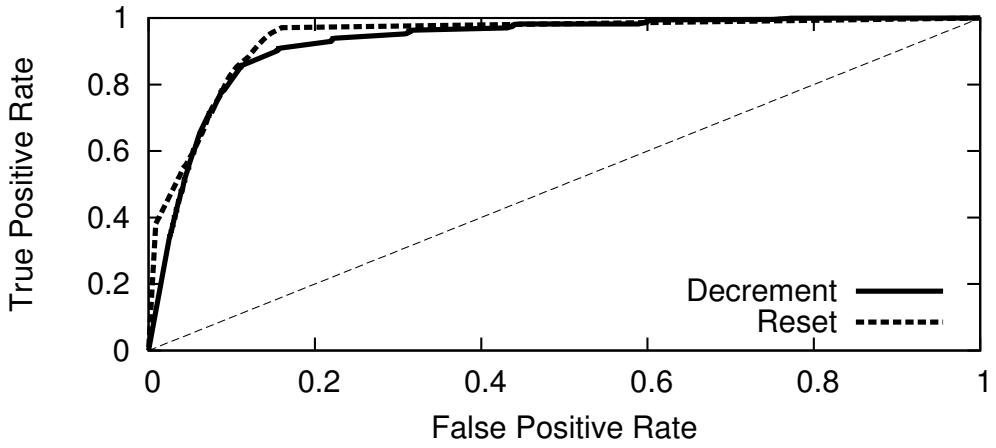

Figure 4.7: ROC curve showing TPR vs. FPR using two SENDME removal strategies: Decrement and Reset. The thin dotted line represents randomly guessing SENDME cells at some rate.

Decrement: We scan through each traffic instance with a running counter of incoming cells. When the counter reaches some amount p_1, the next outgoing cell is classified as a SENDME. When the outgoing cell appears, the counter is decreased by some amount p_2 (the counter can become negative).

Reset: We reset the counter to 0 instead of decreasing it by p_2 after seeing the outgoing cell. One advantage of such an approach, for example, is that amongst a continuous sequence of outgoing cells, only one cell will be considered a SENDME. Without seeing further incoming cells, it is not possible for several adjacent outgoing cells to be SENDMEs, but it is possible for "Decrement" to make such a classification.[1]

We collect data to evaluate this scheme by instrumenting Tor to mark SENDME cells for a 100×40 data set. We present the results as a receiver operating characteristic (ROC) curve in Figure 4.7. We take the false positive rate over all outgoing packets, which is much greater than the total number of SENDME cells. The area under the two curves for Decrement and Reset, as a sum of trapezoids, are 0.926 and 0.940 respectively. The Reset strategy performed somewhat better in our experiments.

We are interested in the points at which $n_{FN} = n_{FP}$, i.e. the total number of false negatives equalled the total number of false positives. This is the point at which the SENDME removal

[1]There is one unlikely exception: both a circuit-level SENDME and a stream-level SENDME are triggered by the same cell, which requires the two counters to match up.

strategy does not change $|P_+|$, the total number of outgoing cells. Since $|P_+|$ is an important feature for classification, we want to keep it the same. For Decrement, this point is at $p_1 = 5$ and $p_2 = 45$, with $a_{TPR} = 0.71$ and $a_{FPR} = 0.076$. For Reset, this point is at $p_1 = 15$, with $a_{TPR} = 0.75$ and $a_{FPR} = 0.082$.

In our previous work [WG13b] we showed that SENDME removal improved classification accuracy for `Wa-OSAD`. Here we test the same strategy for `Wa-kNN`. The attacker applies the same SENDME removal strategy to both his training and testing set. The baseline accuracy is $a_{TPR} = 0.930$ when there is no SENDME removal and $a_{TPR} = 0.936$ when SENDME removal is perfect.[2] We observed $a_{TPR} = 0.931$ for Decrement and $a_{TPR} = 0.928$ for Reset. While perfect SENDME removal would increase the accuracy of classification (for context, a 0.006 increase in a_{TPR} can be considered a 10% decrease in a_{FNR} from 0.070 to 0.064), it appears that our algorithms are not sufficiently effective to improve the accuracy of `Wa-kNN`. We did not apply SENDME removal on the data sets used for other experiments in this work.

4.5.3 Removing Content Noise

We collected two instances of content noise over Tor. First, we downloaded a 10 GB file from a web site that offered speed testing for users. Second, we downloaded 10 minutes of audio from a music site. Considering the number of cells loaded over time for both types of noise, our audio noise is a step function, whereas our file download noise is continuous.

We merged these noise cells with real cells (keeping timing and ordering), and attempted to remove only the noise cells. As in Section 4.5.2, noise cells are positive and real cells are negative, so identifying a real cell as a noise cell would be a false positive event. We attempted two approaches to noise removal: the classification-based approach, and the counting-based approach. In the classification-based approach, the attacker attempts to use machine learning to decide whether a given cell is noise or not. In the counting-based approach, the attacker counts the total number of cells per interval of time and removes a number of noise cells in each interval.

With our noise data, we will find that there are inherent difficulties in both approaches due to the high degree of randomness in noise.

4.5.3.1 Classification-based approach

First, we show a classifier that failed to remove noise accurately, using feature extraction in an approach similar to splitting and WF. Here we mimic the approach of `Pa-FeaturesSVM`:

[2] We remove all SENDMEs and no other cells by using the ground truth from our data set; a realistic attacker cannot do so unless he is the entry guard, in which case he can reliably remove circuit-level SENDMEs.

Table 4.6: Full feature set for classification-based noise removal. We used 65 features. As with Table 4.2, features pertain to a single candidate cell written as P_i.

Number of features	Description
10	$P_{t_{i+k}}$ (intercell time) for k from -4 to 5.
2	The mean and standard deviation of two interpacket times each before and after the candidate cell.
2	The mean and standard deviation of twelve interpacket times each before and after the candidate cell.
2	Total number of outgoing and incoming cells.
49	$P_{\ell_{i+k}}$ (cell direction) for k from -24 to 24.

using an SVM with a feature set. Although we cannot prove that noise removal is *impossible*, at least it is significantly more difficult than splitting and WF. For this experiment, we reduced the file download rate to the web browsing rate, so that the number of cells of each class is about equal.

This problem is similar to split finding, so we will use a similar algorithm: we extract a neighbourhood of cells from each candidate cell, and use a scoring classifier to score the candidate cell. We used 65 features, similar to the features we used for split finding, and we give the full list in Table 4.6. To attempt to classify this set, we used SVM with the radial basis kernel, and parameters $\gamma = 10^{-13}$ and cost for incorrect classification $C = 10^{13}$ to maximize accuracy. We chose SVM because its kernel method may be able to find an implicit feature space where the features are usable, whereas k-NN has no such capability.

There are two classes: noise cells and real cells. With 400 testing and 400 training elements for each, we ran the classifier 100 times on random subsets of the Tor data set. We obtained a TPR of 0.67 ± 0.10 for both types of noise. These values are low compared to what is necessary for accurate page identification after noise removal. Figure 4.6 suggests that at around 20 cells per second (about 24% of web-browsing traffic rate) page identification deteriorates significantly. We found that the TPR has to be at least 0.92 for `Wa-kNN` to achieve a reasonable accuracy. Alternatively, the attacker needs to use a more noise-resilient classification algorithm. We have therefore shown that our attempt at a classification-based approach failed to remove noise.

4.5.3.2 Counting-based approach

We devised two counting-based algorithms, one for removing continuous noise and one for removing step-function noise. We reduced the file download noise rate to the same as the audio

noise rate, so that a comparison could be made between continuous noise and step-function noise. Then, we mixed real cells and noise cells together, as above.

For continuous noise, we calculated the noise cell rate, and attempted to remove the noise by removing 1 incoming cell every t seconds, where $1/t$ is the observed noise rate. More precisely, we divided the noisy cell sequence into portions of t seconds, and removed the first incoming cell from each portion. If there was no cell within a portion, we also attempted to remove an extra incoming cell from the next portion. We can consider this algorithm as a time-based variant of our SENDME removal algorithm, Decrement, in Section 4.5.2.

To remove step-function noise, we used an algorithm that learned the properties of each step, including the average duration of the step, the number of cells in the step, and the amount of time between steps. Then, we removed noise with a counting-based algorithm similar to the one for continuous noise, parameterized by the properties we learned.

We show the results in Table 4.7. The results show very poor accuracy in removing noise, even though we verified that we learned the parameters of the noise correctly: our counting-based algorithms removed significant numbers of real cells. This can be compared to a baseline of 0.5 for file download noise and 0.42 for audio noise if cells were simply removed randomly, at a rate proportional to the total number of real cells and noise cells. Since we learned the parameters of the noise correctly, the resultant number of cells after our faulty noise removal is similar to that of the number of real cells, but the ordering and interpacket timing has changed significantly. Results from both counting-based algorithms are unrecognizable by `Wa-kNN` (accuracy is close to random guessing).

Interestingly, we also found that `Wa-kNN` still had a true positive rate over 0.5 if it was also allowed to train on the results of noise removal. This is similar to the adjusting attacker we investigated in Section 4.5.1. In the worst case, the adjusting attacker would have to prepare a large number of noise data sets for each possible source of noise.

4.5.3.3 Difficulties in noise removal

We identified two reasons why both our classification-based algorithm and our counting-based algorithm failed.

1. **Short-term cell rate variation.** Specifically for file download noise, we found a very large variation in inter-cell time. The mean time was 0.0004 s, but the standard deviation was 0.003 s which is seven times higher. Classification-based algorithms may also be confused by such an inconsistent feature. If we were to use a counting-based algorithm that removes

Table 4.7: Noise removal accuracy for our counting-based algorithm. Noise cells belong to the positive set and real cells belong to the negative set. For example, removing a real cell (believing it was a noise cell) is a false positive.

	TPR	FPR
File download	0.55 ± 0.16	0.39 ± 0.21
Audio	0.66 ± 0.14	0.32 ± 0.12

one cell per 0.0004 s in accordance with the mean noise rate, we would fail to remove a noise cell or incorrectly remove a real cell with high probability.

2. **Long-term cell rate variation.** We also found that there was significant variation in the number of cells sent over time in the long term. We took our noise data and calculated the number of cells sent for every 5 second interval. For file download noise, we observed a mean of 11800 ± 1200 cells per 5 seconds; for audio noise, we observed 540 ± 30 cells per 5 seconds. Both of these variations are high enough that the attacker may not be able to classify the cell rate correctly, thus failing to remove significant amounts of noise.

4.5.4 Removing Noise on Tor Browser

Although our results above show that noise removal is difficult, we believe it is unlikely for Tor users to generate sufficient noise specifically on Tor Browser. The reasons are as follows:

1. *Flash is disabled*. Most multimedia sources require Flash, which is disabled on Tor Browser. On some (but not all) of these pages, a browser pop-up will ask if the user wants to enable Flash. If the pop-up does not appear, the user cannot view the page. Sometimes, even if the user clicks Allow on the pop-up, the page will still fail to load even after refreshing. It is a hassle to load Flash pages on Tor Browser.

2. *Low bandwidth*. Tor nodes offer widely varying bandwidths, which may sometimes make it difficult to load multimedia even at the lowest quality. For example, YouTube suggests a 1,000 kbps connection to view its 360p videos, but Tor circuits often offer less bandwidth [Tor15].

3. *Blocking due to localization*. Some multimedia sites, such as Netflix and Spotify, do not serve users in many countries. Since Tor users appear to originate from their exit node and

thus their exit node's country, these sites may not be accessible on Tor. This is especially problematic for multimedia sites due to copyright issues.

4. *Blocking Tor*. Some sites, such as Twitch, block Tor users flat out by accessing the public list of Tor relays and blocking those IPs. This allows sites to ban misbehaving users, and in recent years this has become a growing problem for Tor [Din14].

Another reason why Tor users might not generate significant noise to affect WF is Tor's circuit dirtiness mechanism. In Tor, circuits more than 10 minutes old (by default) do not accept new TCP connections. Noise that lasts longer than 10 minutes on a single TCP connection (for example, a file download) may cease to be noise at all if the attacker is able to distinguish among different TCP connections. There are two cases where the attacker can distinguish between TCP connections: first, if the client randomly chooses a different entry guard (by default clients have three entry guards to choose from when establishing circuits); second, if the attacker *is* the entry guard currently used by the client.

These issues suggest that the average Tor Browser user is not likely to generate significant noise in their web browsing to adversely affect WF. However, we emphasize that it is not currently possible to obtain data on Tor users to either confirm or deny this statement; we are merely offering arguments to the contrary. Our arguments are tied to the design of Tor Browser and the workings of the Internet, which may change quickly. For example, there is a growing trend of delivering multimedia content with HTML5 instead of Flash; for example, Youtube and Twitch both use HTML5 by default. This may allow Tor users to load multimedia more easily.

4.6 Conclusion and Future Work

In this chapter, we developed several new algorithms to augment website fingerprinting attacks with the ability to counteract detrimental realistic conditions. Such conditions include a stale database, confusion between contiguous packet sequences, and noise. We find that the attacker can indeed update a database and split packet sequences effectively if they are not loaded simultaneously. If packet sequences are loaded simultaneously, however, our splitting algorithms are ineffective; further, we found it very difficult to remove noise at a sufficiently high accuracy to neutralize their effect on classification.

To convincingly demonstrate the realistic accuracy of website fingerprinting, we developed a code base that takes as input any Tor cell sequence, applies time-based and classification-based splitting to obtain page segments, and then applies `Wa-kNN` against a data set to these page segments. This allows the user to examine the accuracy of website fingerprinting quickly.

We are interested in taking this code further, for example, by supporting a browser add-on that attempts to perform website fingerprinting on web pages while the user is visiting them. The end goal is to show that website fingerprinting is realistic.

There is significant future work for investigating the realistic threat of website fingerprinting:

1. Our work in this section has treated the classifier as a black box, but there is indeed hope that classifiers can be made to be more tolerant to noise. In doing so, they would operate better in the noisy real world.

2. We do not have packet sequences from real Tor users. To do so, we would need to log Tor Browser for many Tor users, with their consent. We have not managed to tackle this demanding task.

3. It is certainly possible to improve on the classification algorithms shown in this work. For example, a better algorithm may succeed in removing noise at high accuracy.

4. In this work, the attacker simply seeks to know the web page the client is vistiing; we do not consider what the attacker wants to do with that information, which may include further monitoring, blocking the page, blocking the user, compiling or selling the information, dedicating extra resources to breaking the user's security, and so on. Further optimization to attacker accuracy is possible for attackers with a specific goal. For instance, an attacker who wants to block a page while it is loading would need to consider an attack with high classification accuracy and low classification time on the initial packets of the packet sequence.

While website fingerprinting classifiers are sometimes hampered by harsh conditions, such as large amounts of multimedia noise, in many situations we have seen that the attacks are indeed realistic. Therefore, we consider it necessary to defend clients against website fingerprinting attacks. This opinion is shared by Tor, which has a website fingerprinting defense. We will investigate website fingerprinting defenses in detail in the next chapter.

Chapter 5

Defenses

> "My shield is so sturdy, nothing can pierce it," boasted the weapon peddler. "My spear is so sharp, it can pierce anything."
>
> "But what would happen if you struck your shield with your spear?" one asked.
>
> *Han Feizi*

In Section 3 we developed new website fingerprinting attacks and compared them to previous ones. We found that our newest attack, `Wa-kNN`, is both efficient and effective, and it performs well in the open-world scenario as well. We outfitted these attacks with tools to succeed in the wild in Section 4. In this section, we will develop new website fingerprinting defenses to protect the client. But one wonders: what would happen if we attacked our defenses with our attacks?

In a repeating cycle of ever-improving attacks and defenses, it is the attacker who has the natural advantage. This is because the client must make the first move: if she wants to defend herself against website fingerprinting, she sends data under her best website fingerprinting defense, and the attacker collects the resulting packet sequence. Then, the attacker has any amount of time to figure out the best way to defeat the client's defense and fingerprint the client. Thus, the attacker can choose the best attack for the client's defense. We will show in this chapter that many previous website fingerprinting defenses can be defeated with known attacks that were published after these defenses.

In this chapter, we counter the above phenomenon by developing defenses that cannot be defeated by any website fingerprinting attack. Central to such a defense is the ability to cause packet sequences from two different web pages to look exactly the same: we refer to this as a

collision. In developing such defenses, we answer the spear-and-shield paradox: the shield will hold sturdy. While attacks such as `Wa-kNN` may be able to defeat many defenses, our defense will be effective against any possible attack, including `Wa-kNN`.

5.1 Background

We start our chapter on website fingerprinting defenses with an introduction of its background and setup. In Section 5.1.1, we introduce new terminology to describe and analyze website fingerprinting defenses. Then, in Section 5.1.2, we describe the defense model. We follow up with two metrics we use to analyze every defense: the overhead in Section 5.1.3, and the maximum attacker accuracy in Section 5.1.4. Finally, we characterize known defenses in Section 5.1.5.

5.1.1 Terminology

We denote a defense as D. A defense D takes as input a packet sequence P and returns a *d*efended packet sequence $D(P)$. $D(P)$ is a packet sequence as well, and the terminology for packet sequences applies. For example, the number of outgoing packets after applying D is $|D(P)_+|$.[1]

For now, our terminology fits that of a *deterministic defense* D, which takes in a packet sequence and returns a packet sequence. This work also considers *random defenses* (for example, adding a random number of noise packets), for which D returns a probability distribution over a set of possible packet sequences. We expand our terminology to tolerate random defenses later in Section 5.1.5.3; for now we continue with deterministic defenses.

With abuse of notation, we say that if S is our data set of all packet sequences, then $D(S) = \{D(P) : P \in S\}$ denotes the output after application of the defense on every packet sequence in the set.

Consider outgoing packets for which their packet length $\ell > 0$. We require that

$$\sum_{\hat{\ell}>0:\hat{\ell}\in D(P)_\ell} \hat{\ell} \geq \sum_{\ell>0:\ell\in P_\ell} \ell$$

In other words, $D(P)$ must have at least as much outgoing content as P. Similarly we require that $D(P)$ must have at least as much incoming content as P. We do not allow the defense D to

[1] Note that $D(P_+)$, the defense applied on all outgoing packets of P, is different from $D(P)_+$; the former does not make sense in our context.

drop content from P, which would interfere with the web-browsing client. If all packet sizes are equal, this is simply $|D(P)_+| \geq |P_+|$ and $|D(P)_-| \geq |P_-|$.

Many WF defenses contain several independently functional components, each of which functions as a defense on its own. For two defense components D_{first} and D_{second}, we use the standard function notation $D_{second} \circ D_{first}$ as the defense that applies D_{first} first and D_{second} second.

A defense D needs to confuse the attacker by changing the total amount of information delivered to the user (which is by itself a powerful feature). To do so, D adds *fake packets* (sometimes called dummy packets in the literature) to the packet sequence. Fake packets contain no real information but they are indistinguishable from real packets to the attacker, but they can be identified and dropped by the recipient upon decryption.

5.1.2 Participants of the Defense

In general, it is not sufficient for the defense to be implemented only on the client's side (although it would be desirable).[2] This is because the client has little control over incoming traffic, which the attacker can still use for website fingerprinting. To defend incoming traffic, the client requires the help of a *cooperator*. We refer to the client and the cooperator as the two *participants* of the defense.

From the client's perspective, the cooperator must be located after the expected WF attacker on the network, but before the destination server. On Tor, for example, eavesdroppers on the client's network, the entry node's network, or the entry node itself are possible adversaries. Therefore, cooperators could be the middle node or the exit node of the circuit.

The cooperator has the capability to shape the traffic by adding fake packets or delaying real packets. She may do so according to the client's specifications. In addition, the cooperator drops all fake packets sent by the client before they reach the server. Within the context of website fingerprinting, we assume that the cooperator follows the defense protocol faithfully. We make this assumption because the client can always detect a misbehaving cooperator: a cooperator that delays or adds packets incorrectly will change the defended packet sequence. Furthermore, if the cooperator is the final proxy in the network, she already knows the client's real traffic, so she has no motivation to expose her traffic.

[2] One defense, HTTPOS, is the only known exception; we will discuss it in Section 5.2.1.3.

5.1.3 Overhead

The overhead of a defense is of significant interest to us when designing a defense. If the overhead were not an important factor (for example, if the user controls a powerful proxy and demands complete privacy), it is not hard to design a perfect defense: both the client and the cooperator simply flood the channel to maximum capacity with fake packets at all times. This defense, however, does not suit privacy technologies in general. It is therefore important to minimize overhead while designing defenses, especially because we want to design defenses for all privacy-conscious web-browsing clients.

We divide the overhead of a defense into two components: bandwidth overhead and time overhead.

Bandwidth overhead. The bandwidth overhead $O_D^B(P)$ of a defense D on packet sequence P is the extra number of bytes required to transmit a packet sequence, i.e.

$$O_D^B(P) = \frac{\sum_{\ell_D \in D(P)} |\ell_D| - \sum_{\ell \in P} |\ell|}{\sum_{\ell \in P} |\ell|}$$

In the case of cell sequences, the output $D(P)$ contains cells of the same fixed length as cells in P, so we can write

$$O_D^B(P) = \frac{|D(P)| - |P|}{|P|}$$

Averaging over the data set S, we have

$$O_D^B(S) = \frac{\sum_{P \in S} |D(P)| - \sum_{P \in S} |P|}{\sum_{P \in S} |P|}$$

If D does not remove packets from P, then the bandwidth overhead is simply the ratio of fake cells to real cells.

The bandwidth overhead of a scheme is a burden on the *privacy service*. For example, consider Tor: Tor relies on volunteer nodes who may not be able to serve all of their clients. If their maximum bandwidth capacity is reached, Tor nodes will offer less bandwidth to its clients and delay their traffic. Increasing bandwidth overhead correspondingly decreases the capacity of the network to serve its clients. In turn, this may affect the client's experience as well, as Tor is generally the bottleneck in the client's connection. Often, we are interested in finding the bandwidth-optimal defense (one with the minimum bandwidth overhead) under specific constraints.

Time overhead. The time overhead $O_D^T(P)$ of a defense D on packet sequence P is the extra amount of time required to transmit a packet sequence, i.e.

$$O_D^T(P) = \frac{T_{|D(P)|} - T_{|P|}}{T_{|P|}}$$

We calculate the time overhead separately from the bandwidth overhead: we assume that the network has enough capacity to send the required number of fake cells without delaying packet transmission. If the defense specifies that the fake cells must be delayed by some time before or after real cells, then fake cells would contribute to the time overhead.

The time overhead is a burden on the *client*. A larger time overhead forces the client to wait longer to load web pages, which may be frustrating. This is especially because the time-to-last-byte on Tor is already quite long: loading Alexa's top 100 web pages, we found a mean time-to-last-byte of 16.4 s $\pm$ 6.0 s. A large time overhead may render the network nigh-unusable.

We write both the bandwidth overhead and the time overhead as percentages. For example, a client using a defense with 100% bandwidth overhead and 200% time overhead expects to send twice as many packets in thrice as much time as a client who is not using any defense (0% bandwidth and time overhead).

5.1.4 Maximum Attacker Accuracy

In many previous website fingerprinting works on defenses, researchers tested their defenses against specific WF attacks [WCM09, LZC+11, DCRS12] to show their effectiveness. However, other researchers then found new WF attacks that can defeat those defenses [CZJJ12, WG13b, WCN+14]. We give concrete evidence for this phenomenon in our survey of previous work in Section 5.2. The client cannot fully trust these defenses to carry privacy-sensitive traffic, especially against invested adversaries that are willing to update their attacks. In this work, we want to find provably effective defenses that hold up against all website fingerprinting attacks.

For a defense D to be provably effective against all possible attacks, the attacker must be unable to distinguish between the packet sequences of two different web pages. Mathematically, for two different packet sequences ($P \neq P'$) that belong to different classes ($c(P) \neq c(P')$), suppose defense D satisfies:

$$D(P) = D(P')$$

We call such an event a *collision* between P and P'. The two packet sequences must be equal in time, direction, and length of all packets. Collisions are central to our analysis of defense effectiveness. We define $S_{col}(D(P)) \subseteq S$, the *collision set* of P, as:

$$P, P' \in S_{col}(D(P)) \implies D(P) = D(P')$$

We say that D collides $S_{col}(D(P))$.

To analyze defenses we consider a *perfect attacker*. The perfect attacker uses the best information-theoretic classification strategy that gives the maximum possible accuracy, which we refer to as the maximum attacker accuracy (MAA). 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. We can also define the perfect attacker as one for which the training and testing sets are the same, and the attacker simply uses a lookup table to classify packet sequences.

The perfect attacker can only fail to classify when there is a collision. In this case, even the perfect attacker cannot correctly determine the web page. Consider the collision set $S_{col}(\hat{P}) = \{P_1, P_2, \ldots, P_n\}$, i.e. $D(P_i) = \hat{P}$ for $1 \leq i \leq n$. Denote the set $S_{col}^{C_i}(\hat{P})$ as:

$$S_{col}^{C_i}(\hat{P}) = \{P_i \in S_{col} : c(P_i) = C_i\}$$

The attacker's best strategy is to guess C_{max} for which:

$$C_{max} = \underset{C_i}{\operatorname{argmax}} |S_{col}^{C_i}(\hat{P})|$$

In other words, the attacker guesses the class that contains the most packet sequences in $S_{col}(\hat{P})$. The MAA of the set $S_{col}(\hat{P})$ is then

$$a^M(S_{col}(\hat{P})) = |S_{col}^{C_{max}}(\hat{P})|/|S_{col}(\hat{P})|$$

This is easy to see: the number of correct guesses in the set $S_{col}(\hat{P})$ is $|S_{col}^{C_{max}}(\hat{P})|$.

The MAA of the union of two different collision sets $S_{col}(\hat{P})$ and $S_{col}(\hat{P'})$ is the weighted average over the two sets:

$$a^M(S_{col}(\hat{P}) \cup S_{col}(\hat{P'})) = \frac{|S_{col}(\hat{P})| \cdot a^M(S_{col}(\hat{P})) + |S_{col}(\hat{P'})| \cdot a^M(S_{col}(\hat{P'}))}{|S_{col}(\hat{P})| + |S_{col}(\hat{P'})|}$$

We can then extend this definition to apply to the entire data set S. When we give the MAA of a defense D, we always apply it to the entire data set S we collected.

The maximum attacker accuracy can be considered the effectiveness of a defense, and the overhead can be considered its efficiency. In all of our defenses, it is possible to trade a lower effectiveness for greater efficiency and vice versa.

Example 4. Suppose we have a set of packet sequences $S = \{P_1, P_2, P_3, P_4, P_5\}$, with $c(P_3) = c(P_4) = c(P_5) \neq c(P_1)$. Suppose there is only one collision set, S itself, such that $P_i, P_j \in S \implies D(P_i) = D(P_j) = \hat{P}$. We have $a^M(S) = 3/5$: the perfect attacker will guess $c(P_3)$ always. Whether $c(P_1) = c(P_2)$ or not does not change the perfect attacker's strategy, nor does it change the MAA.

Suppose we have another set for which $S' = \{P'_1, P'_2, P'_3\}$ and $c(P'_2) = c(P'_3) \neq c(P'_1)$, and similarly only one collision set after applying D, being S' itself. Then $a^M(S') = 2/3$, and we have $a^M(S \cup S') = (2+3)/(3+5) = 5/8$.

We write the number of collision sets in S as $n_{col}(S)$. n_{col} is related to a^M: intuitively, the greater the number of collision sets, the smaller each collision set is, and the greater the maximum attacker accuracy. Often, $n_{col}(S)$ is a parameter in defenses, since the defense participants can determine $n_{col}(S)$ but cannot determine $a^M(S)$: $a^M(S)$ may be different between the testing and training sets, but $n_{col}(S)$ is not likely to change. We evaluate the MAA of defenses on the same data sets we collected in Section 2.4.

5.1.4.1 Open-world analysis

The open-world scenario is interesting to us as well, for the same reason it is interesting for attacks: a high false positive rate could significantly deter a potential attacker, even the perfect attacker. However, the maximum attacker accuracy is not well defined in the open-world scenario. This is because the attacker could trade a^{MTPR} (the maximum TPR) for a^{MTNR} (the maximum TNR) and vice-versa by respectively biasing classification towards or away from the non-monitored set. Such bias could itself be dependent on the properties of the collision set; the perfect attacker's strategy is now indeterminate.

We give one example of such a biased strategy. When classifying a collision set, the attacker classifies according to the non-biased method with probability p (choosing the most commonly occurring class), and the attacker simply classifies it as the non-monitored class with probability $1 - p$. If the non-biased method had TPR and FPR values of a^{MTPR} and a^{MFPR}, this method would have $a^{MTPR} \cdot p$ and $a^{MFPR} \cdot p$.

In this work, we will perform experiments in the open world, assuming that the attacker has no such bias. a^{MTPR} and a^{MTNR} are thus given for an attacker with no bias. When there is no bias, the attacker's total number of true classifications (true positives plus true negatives) is maximal. The reader should note that the attacker can trade a decreased number of true positives for an increased number of true negatives (though the increase in the number of true negatives will be less than the decrease in the number of true positives), and vice-versa.

Optimality. We say that a defense D is no worse than D' on S if $O_D^B(S) \leq O_{D'}^B(S)$, $O_D^T(S) \leq O_{D'}^T(S)$ and $a_D^M(S) \leq a_D^M(S')$, and we say that D is better than D' if at least one of those terms is not equal.

We can phrase the objective of finding a defense D as follows:

For some data set S, find D that minimizes $O_D^B(S)$, $O_D^T(S)$, and $a_D^M(S)$.

Often, we do not allow the participants of the defense to know S; they must decide D without knowing the exact packet sequences of S. We will discuss the effect of different levels of information on S in Section 5.3.

5.1.5 Defense Characterization

5.1.5.1 Simulatable and non-simulatable defenses

We define the following *simulatable operations* on a packet sequence P:

1. Add a fake packet to P. A fake packet contains no real content; it can be understood as a packet of encrypted zeroes.
2. Delay a packet.
3. Split a packet into two packets, such that their contents add up to that of the original.
4. Pad a packet with encrypted zeroes, so that it appears longer than it is.

Simulatable operations are modifications to the packet sequence that cannot remove or re-order any real content from the sequence. On Tor cell sequences, we do not need to split or pad cells, so that the only simulatable operations are adding a cell or delaying a cell. We define a simulatable defense as one that only has simulatable operations, and a non-simulatable defense as one that also has other operations. Since non-simulatable defenses cannot remove real content either, the difference is that non-simulatable defenses can re-order packets. For example, adding fake packets as noise to confuse an attacker is a simulatable defense. On the other hand, increasing the maximum number of simultaneous connections in the web browser (see our discussion of simultaneous connections in Section 2.1.1) is a non-simulatable defense (though it is not an effective defense).

The effect of a simulatable defense on packet sequences can be simulated without collecting a new packet sequence. Simulatablility is thus a desirable property for design as we can test simulatable defenses quickly and repeatedly over a single data set for optimization.

Implementation-wise, since the cooperator cannot read the client's packets, she cannot drop or re-order any real packets for fear of scrambling real content; she can only implement simulatable operations. This implies that only the client can implement non-simulatable defenses. Indeed, all known non-simulatable defenses involve some modification of the client's browser.

5.1.5.2 Limited and general defenses

In Section 5.1.4, we said that D collides packet sequences P and P' when

$$D(P) = D(P')$$

To classify known defenses for a clearer comparison, we say that a defense D is *limited* if it does not cause collisions between packet sequences and D is *general* otherwise. The distinction between the two is clear-cut for known website fingerprinting defenses: we consider the defense limited if we cannot observe collisions in our website fingerprinting data set with 20,000 elements.

General defenses seek to cause collisions; from another perspective, general defenses seek to cover all packet sequence features between different packet sequences. Known general defenses are similar in their approach to do so. We list three common components in known general defenses, in terms of the feature sets in Section 3.3:

Packet padding (D_{ppad}): Adding content-less bytes to the end of packets, so that all packets have the same length. Packets that are already longer than this length will be split. This covers unique packet lengths. Tor uses packet padding, which is effective against many older attacks.

Packet scheduling (D_{sched}): Fixing the times at which participants of the defense can and must send packets. If each participant has no real content to send, she sends dummy packets. This covers packet ordering and interpacket timing.

Sequence padding (D_{spad}): Adding content-less packets to packet sequences. This covers sequence lengths.

These components are implemented in order, so that a general defense D is:

$$D = D_{spad} \circ D_{sched} \circ D_{ppad}$$

We give a brief description of each component in the following.

Packet padding (D_{ppad}). We want to cover unique packet lengths because it is a useful website fingerprinting feature. Packet padding simply pads each packet less than some size ℓ_{ppad} so that it has size ℓ_{ppad}. With encryption, the extra packet padding cannot be distinguished from real content, so the true length of the packet is hidden.

As described in detail in Section 2.1.2, Tor performs packet padding by default by delivering content in fixed-size 512-byte cells. While the TCP packets in the end may have different sizes since several Tor cells can be packaged in one TCP packet, the objective of obscuring the true packet length is nevertheless achieved.

Sometimes, we will use two packet sizes ℓ_{ppad+} and ℓ_{ppad-}, to pad outgoing packet sizes and incoming packet sizes respectively. The attacker can already distinguish between outgoing and incoming packets; padding them to different sizes will not affect the MAA.

Packet scheduling (D_{sched}). Packet scheduling dictates when packets can and must be sent. Let Z_{sched} be a list of elapsed times in ascending order. We say that a defense applies packet scheduling according to Z_{sched} if each participant sends a packet at every time $T \in Z_{sched}$. If there is no real content to send at time T, each participant sends a fake packet instead.

We can have two lists of times, Z_{sched+} and Z_{sched-}, dictating the timing of outgoing and incoming packets. Otherwise, Z_{sched} will also specify the direction of each packet. Note that Z_{sched} may be infinite; in fact, we prefer Z_{sched} to be infinite so as to avoid the case when the sequence lengths are not padded (see our description of BuFLO in Section 5.2.2.1).

Packet scheduling using Z_{sched} leaves only the sequence length as a feature for the attacker. In other words, $|D_{sched} \circ D_{ppad}(P)| = |D_{sched} \circ D_{ppad}(P')| \implies D_{sched} \circ D_{ppad}(P) = D_{sched} \circ D_{ppad}(P')$. $D_{sched} \circ D_{ppad}(P)$ would be exactly the first $|D_{sched} \circ D_{ppad}(P)|$ packets of Z_{sched}. If we separate outgoing and incoming packets, then packet scheduling leaves the outgoing sequence length and incoming sequence length as the only two features for the attacker.

Packet scheduling removes both packet ordering and interpacket timing as features. Intuitively, the occurrence of a packet has no relation to web page content, as its occurrence was dictated by Z_{sched} before the page was loaded. However, packet scheduling can cause a large time overhead, as all packets must be delayed until the next permitted time $T \in Z_{sched}$.

Sequence padding (D_{spad}). Much as packet padding covers unique packet lengths, sequence padding covers the length of the sequence (the total number of packets). There are two ways to do so: deterministically, and randomly.

Deterministic padding is done with a process we refer to as *rounding*. When rounding a sequence length, we add packets until the sequence length reaches one out of a set of positive

integers; we refer to the set as Z_{spad}. For deterministic sequence padding schemes, Z_{spad} completely determines D_{spad}.

Formally, we define a rounding function $round(a, Z_{spad})$, where a is an integer we want to pad and Z_{spad} is a set of positive integers we call the rounding set. We define $round(a, Z_{spad})$ as follows:

$$round(a, Z_{spad}) = \begin{cases} \min\{z \in Z_{spad} : z \geq a\} & \text{if } \exists\, z \in Z_{spad} : z \geq a \\ a & \text{otherwise.} \end{cases}$$

For example, if Z_{spad} contains all multiples of 50, then the effect of $round(a, Z_{spad})$ is to round a up to the nearest multiple of 50. When adding packets, we follow the packet scheduling dictated by Z_{sched}. Much like Z_{sched}, it is preferable that Z_{spad} be infinite.

Sometimes, we round outgoing sequence length and incoming sequence length separately. We define two rounding sets Z_{spad+} and Z_{spad-}, and we pad the sequence lengths $|P_+|$ and $|P_-|$ to $round(|P_+|, Z_{spad+})$ and $round(|P_-|, Z_{spad-})$ respectively by adding fake outgoing and incoming packets. Fake packets contain no real content (e.g. encrypted zeroes) and they are dropped by both participants when received.

For random padding, let us denote a probability distribution over the non-negative integers as $Z^r_{spad} : \mathbb{Z}_{\geq 0} \to [0, 1]$. To pad packet sequence P, we draw some z from Z^r_{spad}, and add z packets to the end of P. Similar to our strategy with deterministic padding, we will have Z^r_{spad+} and Z^r_{spad-} to pad outgoing and incoming packets respectively, and adding packets is subject to packet scheduling.

We note that while sequence padding is done at the end of the packet sequence, it is non-trivial for the cooperator to know when to start sequence padding for Z_{sched-}, as she does not know when the data stream has ended. One way for the proxy to know this is to set a parameter t_{end}, such that if there were no real packets in the last t_{end} seconds, we determine that page loading has ended.[3] A premature sequence padding only somewhat increases the overhead, so its selection is not significantly interesting.

Deterministic and random sequence padding have a very similar effect, which we show in the following example:

Example 5. Let us consider two sequence padding mechanisms, D using a deterministic rounding set Z and D^r using a probability distribution Z^r. Z consists of all multiples of 50, while Z^r

[3] The reader may note that this is quite similar to time-based splitting in Section 4.3.3. However, incorrectly deciding that a page load has ended significantly degrades time-based splitting, but has little effect on the quality of sequence padding.

picks a number randomly and uniformly between 0 and 49 and adds that number of fake packets. On average, Z and Z^r both expect to add 24.5 packets: their bandwidth overheads are the same.

Consider the collision set of some packet sequence P with $|P| = 80$ after packet scheduling and padding. For deterministic padding with a rounding set, we have $|D(P)| = 100$. The collision set of P consists of all packet sequences in $P' \in S$ which satisfy $50 < |P'| \leq 100$. From the attacker's perspective, seeing that $|D(P)| = 100$, the attacker only knows that $50 < |P| \leq 100$.

For random padding with the uniform distribution, let us suppose that between 0 and 50, Z^r happened to select some L_r. The result is $|D^r(P)| = 80 + L_r$, and the collision set of P consists of all packet sequences $P' \in S$ that satisfy $30 + L_r < |P'| \leq 80 + L_r$. If L_r happens to be 20, then the collision set is exactly the same as the deterministic case.

Often, we have a tradeoff between the bandwidth overhead and the time overhead, which is a tradeoff between burdening the service and burdening the client. We will discuss this tradeoff in detail while evaluating each defense.

In describing each general defense, we will analyze each of the above components. We will use the above terminology, describing ℓ_{ppad}, Z_{sched}, and Z_{spad}, allowing an easy comparison between the defenses.

5.1.5.3 Deterministic and random defenses

We saw two different mechanisms for sequence padding: padding to a rounding set, which always gives the same output for the same input; or adding a random number of fake packets, which can give different outputs for the same input. We refer to the former as a *deterministic* defense and the latter as a *random* defense. In defining MAA and collision sets we assumed a deterministic defense; here, we discuss those terms for a random defense.

When a random defense D^r is applied to a sequence $P \in S$, the result $D^r(P)$ is now a discrete probability distribution over possible output sequences. We write $D^r(P)$ as the possible output packet sequences of D^r applied on P, such that $D^r(P)_i$ is a packet sequence for all $1 \leq i \leq |D^r(P)|$; the probability of D^r on P outputting $\hat{P}$ is written as $Pr_{D^r(P)}(\hat{P})$. We do not write the subscript if the context is clear. In this case, we need to calculate the MAA differently.

Suppose the attacker sees defended packet sequence $\hat{P}_{real} \in D(P_{real})$, where $D(P_{real})$ is the defense applied on P_{real}. Our perfect attacker knows D, but he is not be able to remove dummy packets trivially; that is to say, the attacker cannot extract P_{real} from $\hat{P}_{real}$. Instead, the attacker proceeds as follows. The attacker correctly computes $Pr_{D(P)}(\hat{P}_{real})$ for all $P \in S$; for example,

the attacker certainly knows that $Pr_{D(P_{real})}(\hat{P}_{real}) > 0$. Similar to the deterministic case, the attacker's optimal strategy is to guess the class with the highest likelihood of generating $\hat{P}_{real}$.

We denote $S^r_{col}(\hat{P})$ as the set of all elements in S that may map to $\hat{P}$, analogous to the collision set for deterministic defenses. However, unlike S_{col}, S^r_{col} does not partition S: the same packet sequence in S can appear in $S^r_{col}(\hat{P})$ for many $\hat{P}$. When seeing $\hat{P}$, the perfect attacker chooses to guess the class that occurs the most frequently, which we denote as C_{max} (determined by computing $Pr_{D(P)}(\hat{P})$ for all packet sequences). Then the attacker's maximum accuracy over $\hat{P}$ is:

$$a^M(S^r_{col}(\hat{P})) = \frac{\sum_{P \in C_{max}} Pr_{D(P)}(\hat{P})}{\sum_{P \in S} Pr_{D(P)}(\hat{P})}$$

The above is the accuracy of the collision set that maps to $\hat{P}$. As for deterministic defenses, $a^M(S)$ averages over all possible collision sets on S, multiplied by the occurrence probability of each collision set. However the set of all possible collision sets is often prohibitively large for random defenses. When we evaluate a defense D for which we cannot computationally exhaust all possibilities, we apply D multiple times to each element $P \in S$ and perform the calculation as above. For our experiments in Section 5.5.5, we will apply D 10,000 times in total to obtain small 95% confidence intervals.

5.1.6 Theory on General Defenses

In Section 5.1.4 we positioned the problem of searching for a good general defense as an optimization problem: find D that minimizes the maximum attacker accuracy (a^M_D), bandwidth overhead (O^B_D) and time overhead (O^T_D). In this section we present theoretical results on optimality for general defenses, allowing us to search for optimal defenses efficiently. Our results here justify our construction for general defenses, which we describe in Section 5.2.2.

We proceed with the assumption that all packet sequences in S come from different classes, i.e. $P \neq P' \rightarrow c(P) \neq c(P')$. This is a conservative assumption when analyzing general defenses. In terms of the maximum attacker accuracy, it is equivalent to assuming that all web pages always consistently produce the same packet sequence, which is most optimistic for the attacker.

5.1.6.1 Maximum attacker accuracy

First, let us consider the maximum attacker accuracy (MAA) of deterministic defenses, a^M. On any data set S, no classifier can achieve an accuracy above $a^M(S)$.

Lemma 3. For a deterministic defense, with the MAA denoted as a^M and the number of collision sets in S denoted as $n_{col}(S)$, we have

$$a^M(S) = \frac{n_{col}(S)}{|S|}$$

Proof. Since collision sets partition a data set, we can write S as a union of its distinct collision sets:

$$S = S_1 \cup S_2 \cup \ldots \cup S_{n_{col}(S)}$$

One implication is that

$$\sum_{i=1}^{n_{col}(S)} |S_i| = |S|$$

For each collision set $S_i \subseteq S$, since the class of every element in S is different, the attacker has no better strategy than to guess that each element in S_i belongs to a class which appears only once, so we have

$$a^M(S_i) = \frac{1}{|S_i|}$$

The MAA on S is the weighted average of the MAA on each collision set:

$$\begin{aligned} a^M(S) &= \frac{\sum_{i=1}^{n_{col}(S)} |S_i| \cdot a^M(S_i)}{\sum_{i=1}^{n_{col}(S)} |S_i|} \\ &= \frac{\sum_{i=1}^{n_{col}(S)} |S_i|/|S_i|}{|S|} \\ &= \frac{n_{col}(S)}{|S|} \end{aligned}$$

□

Lemma 3 is significant in our search for optimal defenses. The number of collision sets directly determines the MAA: the exact composition of each collision set does not matter. If we were to remove one element from a collision set and add it to another, then the decreased MAA

of the expanded collision set would exactly compensate for the increased MAA of the shrunken collision set.

At the start of the section we started with the assumption that $|C_i| = 1$ for all classes. Lemma 3 can in fact be expanded to the case where $|C_i| = |C|$ for all classes:

$$a^M(S) \leq \frac{n_{col}(S)}{|S|/|C|}$$

Nevertheless we will continue with the case for $|C| = 1$.

Next we consider random defenses. We computed the MAA of random defenses in Section 5.1.5.3. Unfortunately, for random defenses there is no clear connection between the number of collision sets and the MAA. We illustrate this fact with the following example:

Example 6. Suppose we have already performed packet padding and scheduling, and packet scheduling leaves only the sequence length as a feature, so we are specifically considering random sequence padding. We have data set $S = \{P^1, P^2\}$ where $|P^1| = 100$ and $|P^2| = 200$. We define D^r as $D^r(P^1) = \{P^1, P^2\}$ at probabilities $Pr_{D^r(P^1)}(P^1) = 0.5$ and $Pr_{D^r(P^1)}(P^2) = 0.5$, while $D^r(P^2) = \{P^2\}$ (no change). Then, $O^B_{D^r}(S) = 25/150 = \frac{1}{6}$ (there is only an overhead when $D^r(P^1) = P_2$), and $MAA(S) = \frac{3}{4}$ (the attacker guesses incorrectly only when $D^r(P^1) = P^2$).

Note that there are only two possible optimal deterministic defenses on this set: no rounding with $O^B_D(S) = 0$ and $MAA(S) = 1$, and a rounding set of $Z_{spad} = \{200\}$ where $O^B_D(S) = \frac{1}{3}$ and $MAA(S) = \frac{1}{2}$. However neither is strictly better than the above D^r. This suggests that random defenses fill a niche between the two extremes.

Now consider that $Pr_{D^r(P^1)}(P^1) = 1 - x$ (instead of 0.5), and $Pr_{D^r(P^1)}(P^2) = x$. We have $O^B_{D^r}(S) = \frac{x}{3}$ and $MAA(S) = 1 - \frac{x}{2}$. Since $\frac{1}{2}$ is the minimum possible MAA with two packet sequences, we see that having two collision sets for a random defense tells us no information at all about the MAA: it can range from the minimum to the maximum depending on x.

Another implication of the above is the deterministic defenses do not dominate random defenses. There are random defenses for which no deterministic defense can achieve both a better MAA and overhead. We discuss the relationship between deterministic and random defenses further in Section 5.1.6.3, where we investigate sequence padding. We proceed to investigate theoretical results on the general defense components described in Section 5.1.5.2: packet scheduling and sequence padding.

5.1.6.2 Packet scheduling

Here we consider how we can optimize packet scheduling to minimize the overhead. Suppose we have performed packet padding (e.g. we start with Tor cell sequences), so that packets of all packet sequences have the same length. We say that packet sequence P is a *supersequence* of packet sequence Q (or Q is a subsequence of P) if there exists a mapping $f : Z^+ \to Z^+$ from indices of packets in Q to indices of packets in P such that f satisfies the following two properties:

1. f is strictly monotonic: for two packets Q_i and Q_j with $i > j$, we have $f(i) > f(j)$.
2. f cannot map packets earlier: For the i-th packet of Q occurring at elapsed time T_{Q_i}, we have $T_{P_{f(i)}} \geq T_{Q_i}$.

We can envision f as being analogous to Z_{sched} in scheduling when the real packets of Q should be sent alongside fake packets to collide Q with P.

Recall our definition of simulatable defenses in Section 5.1.5.1. Here we consider simulatable operations on packet sequences where all packets have the same length, so the only simulatable operations are adding fake packets or delaying real packets. Our definition of simulatable defenses can then be rephrased in terms of supersequences: for any defense D and all packet sequences P, $D(P)$ must be a supersequence of P. The reader can verify that those two definitions are equivalent.

Recall that the bandwidth overhead of applying D to S is

$$O_D^B(S) = \frac{\sum_{P \in S} |D(P)|}{\sum_{P \in S} |P|} - 1$$

Given S, we want to identify D such that $O_D^B(S)$ is minimal.

Now let us consider $O_D^B(S_{col}(P))$, the bandwidth overhead of the collision set of P. Since $D(P)$ is the same for all elements in $S_{col}(P)$, we have

$$O_D^B(S_{col}(P)) = \frac{|D(S_{col}(P))| \cdot |S_{col}(P)|}{\sum_{P \in S} |P|} - 1$$

We can therefore see that minimizing O_D^B over $S_{col}(P)$ is simply a matter of choosing the shortest possible $D(S_{col}(P))$. For a simulatable D we have seen that $D(S_{col}(P))$ must be a supersequence of all $P \in S_{col}(P)$. Thus, the optimal packet scheduling scheme for a simulatable defense is the *shortest common supersequence (SCS)*. Finding the SCS is a well-studied NP-hard problem and we will consider it in Section 5.3.

5.1.6.3 Sequence padding

All known general defenses apply sequence padding after packet scheduling. For simplicity, let us consider packet scheduling using Z_{sched} that leaves only the sequence length available for the attacker. Thus, packet sequences in this subsection satisfy $|P| = |P'| \implies P = P'$.

In (deterministic) sequence padding, the rounding set $Z_{spad} = \{z_1, z_2, \ldots\}$ partitions a data set S. We define $S_i \subseteq S$ as the subset of S for which $P \in S_i \implies D(P) = z_i$. For simplicity of notation, we thus write $D(S_i) = z_i$. We allow S_i to be empty for some i; no packet sequences collide to that z_i.

Note that the connection between the time overhead and bandwidth overhead has been determined by packet scheduling. Sequence padding only changes the total number of packets, so it is only necessary to consider the total number of packets added, which determines the time overhead of sequence padding. Thus, we restrict our attention to the bandwidth overhead when considering sequence padding. The bandwidth overhead of the rounding set Z_{spad} on data set S is

$$O_D^B(S) = \frac{\sum_{i=1}^{m} |S_i| z_i}{\sum_{P \in S} |P|} - 1$$

Optimizing O_D over S is thus equivalent to minimizing $\sum_{i=1}^{m} |S_i| z_i$.

We start with the following result:

Lemma 4. Averaging across all possible sets of packet sequences under some length M, any overhead-optimal deterministic sequence padding defense D_{spad} must use an arithmetic sequence rounding set, where:

$$Z_{spad} = \{L_{seg}, 2L_{seg}, 3L_{seg}, \ldots\}$$

Proof. We average across all possible sets of packet sequences so that we have a uniform distribution of packet sequence lengths across the range.

Denoting $\Delta z_i = z_i - z_{i-1}$, with $z_0 = 0$, the probability of a packet sequence satisfying $D_{spad}(P) = z_i$ is equal to $\Delta z_i / M$. On average, when rounding to z_i, the bandwidth overhead is $\Delta z_i / 2$ since the distribution is uniform. Thus the mean amount of padding for any packet sequence is

$$O_D^B = \frac{\sum (\Delta z_i)^2}{2M}$$

The problem of optimizing the bandwidth overhead is therefore:

$$\begin{array}{ll} \text{minimize} & \sum(\Delta z_i)^2 \\ \text{subject to} & \sum \Delta z_i = M \end{array}$$

Note that M is a constant. It is easy to see that the solution for the above procedure is when $\Delta z_i = \Delta z_j$ for all i, j, which is the arithmetic sequence. □

Specifically, if there are m collision sets, the bandwidth overhead of the arithmetic sequence is simply $1/m$. Consider the above scenario, where packet sequence lengths are uniformly distributed, and supposing the largest packet sequence in our data set is L_{max} (for simplicity, we assume L_{max} is a multiple of L_{seg}) and the smallest is 0, the average packet sequence length is $L_{max}/2$. With an arithmetic sequence padding to the next multiple of L_{seg}, the mean amount of packets added is $L_{seg}/2$. Thus, the bandwidth overhead would be L_{seg}/L_{max}: it increases linearly with L_{seg}.

It should be noted that while packet sequence lengths are not likely to be uniformly distributed in a client's set of visited web pages, this assumption is suitable when the defense has no knowledge of the client's packet sequence length distribution. Even when the defense has specific knowledge of this distribution, we would not want to leak such information through the parameters of the defense, which are easily observable by the attacker. Lemma 4 can be adapted for other general classes of distributions; we give a general algorithm to find overhead-optimal rounding sets given a specific input set of packet sequences in Section 5.4.5.

Since Lemma 3 tells us that the choice of collision sets does not affect the MAA, Lemma 4 tells us that arithmetic sequence rounding sets achieve a form of optimality: they are optimal in both MAA and overhead when averaged over all possible packet sequence sizes. Note that Lemma 4 does not hold when there is a prior distribution on packet sequence sizes. Nevertheless there is a strong incentive to use arithmetic sequence rounding sets, given that realistically we cannot collect information on the set of sites the client wants to visit.

Cai et al. [CZJJ12] considered an alternative general defense that used geometric sequence rounding sets, where $Z_{spad_i} = L_{base}^i$, with i acting as the exponent. For packet sequences whose scheduled lengths are uniformly distributed under some L_{max} (the same scenario as in Lemma 3), it is easy to see that the mean bandwidth overhead is $(L_{base} - 1)/2$, whereas the number of collision sets is $\log L_{max} / \log L_{base}$. With the same number of collision sets, the mean bandwidth overhead for arithmetic sequence rounding sets is

$$O_D^B = \frac{\log L_{base}}{2 \log L_{max}}$$

The arithmetic sequence overhead is strictly smaller than the geometric sequence overhead. We can see this by observing that $\log L_{base}$ grows more slowly than $L_{base} - 1$ with increasing L_{base}, and they are equal at the minimum possible L_{base}, where it is equal to 1 (no padding). As an example, if $L_{max} = 4096$, $L_{base} = 2$, then there are 12 collision sets, the overhead of the geometric sequence rounding set is 50%, and the overhead of the arithmetic sequence rounding set is $1/24 \approx 4.2\%$.

In Section 5.4, we will show a deterministic algorithm to find the bandwidth-optimal rounding set for any data set. This algorithm is computationally feasible for small values of n_{col} (and thus small values of the MAA).

Next we investigate random defenses, and here we consider the *information gain* of a perfect attacker. The attacker may have some prior on the distribution of packet sequences that he expects to observe from the client. Recall that we assumed the sequence length is the only remaining feature, such that guessing $|P|$ from $D(P)$ is the best the attacker could do (and could still lead to de-anonymization). In other words, the attacker attempts to learn

$$Pr\Big(|P|\Big||D(P)|\Big)$$

This Bayesian formulation has the advantage that we tolerate an attacker's prior information $Pr(|P|)$ on the packet sequence lengths that he expects to observe. We use this formulation to investigate random defenses.

We consider a random sequence padding mechanism which randomly adds a number of packets from some distribution Z^r to any input packet sequence, ignoring the input packet sequence itself. So the defense D^r can be written as

$$|D^r(P)| = |P| + x$$

where x is sampled from Z^r. We investigate the class of such defenses:

Lemma 5. Suppose random defense D^r draws from distribution Z^r, and samples from Z^r range from m_{min} to m_{max}. For all such distributions Z^r, the uniform distribution gives the attacker the minimum amount of information.

Proof. Any distribution Z^r drawing from $[m_{min}, m_{max}]$ will let the attacker know that

$$|D(P)| - m_{max} \leq |P| \leq |D(P)| - m_{min}$$

We want to show that the uniform distribution gives no further information.

The defense D defines a distribution such that

$$Pr\big(|D(P)|\big||P|\big) = Pr\big(Z^r = |D(P)| - |P|\big)$$

From Bayes' rule, we have

$$\begin{aligned} Pr\big(|P|\big||D(P)|\big) &= \frac{Pr\big(|P|\big)Pr\big(|D(P)|\big||P|\big)}{Pr\big(|D(P)|\big)} \\ &= \frac{Pr\big(|P|\big)Pr\big(Z^r = |D(P)| - |P|\big)}{\sum_L Pr\big(L\big)Pr\big(Z^r = |D(P)| - L\big)} \end{aligned}$$

For the uniform distribution we have

$$Pr\big(Z^r = L - |P|\big) = \begin{cases} \dfrac{1}{m_{max} - m_{min}} & \text{if } m_{min} \leq L - |P| \leq m_{max} \\ 0 & \text{otherwise.} \end{cases}$$

So the attacker will learn

$$Pr\big(|P|\big||D(P)|\big) = \frac{Pr\big(P\big)}{\sum\limits_{L=|D(P)|-m_{max}}^{|D(P)|-m_{min}} Pr\big(L\big)}$$

Thus, the attacker gains no information except that P is within $[|D(P)|-m_{max}, |D(P)|-m_{min}]$, which is the minimum amount of information. □

The uniform distribution only allows an attacker to "zoom in" to a range of sequence lengths to make his guess, but gives no further information. This is very similar to the effect of deterministic rounding sets, which also give no further information except a possible range of lengths. We formalize this observation as follows:

Lemma 6. Consider a data set S of packet sequences with sequence lengths less than some L_{max} after packet scheduling.[4] Consider defense D applying an arithmetic sequence rounding set that rounds packet sequences to multiples of L_{seg}, and a defense D^r that adds 0 to L_{seg} packets uniformly at random. Both of these defenses have worst-case MAA:

$$a_D^M = a_{D^r}^M = \frac{L_{max} + L_{seg}}{|S| \cdot L_{seg}}$$

[4] The distribution of packet sequence lengths in S has no restriction.

Proof. The arithmetic sequence rounding set MAA can be derived directly from Lemma 3, where we had

$$a_D^M(S) = \frac{n_{col}(S)}{|S|}$$

We have

$$n_{col}(S) \leq (L_{max} + L_{seg})/L_{seg}$$

This is because all packet sequences P such that $|P| \in [mL_{seg} + 1, (m+1)L_{seg}]$ must belong to the same collision set.

For D^r, using a uniform distribution, consider the set of all possible output sequences, which are sequences in the range of $[0, L_{seg} + L_{max}]$. For some particular length L, suppose its collision set is $n_{col}(L)$. We have:

$$n_{col}(L) = \{P \in S : L - L_{seg} \leq |P| \leq L\}$$

The occurrence probability of L is

$$|n_{col}(L)| \cdot \frac{1}{L_{seg} \cdot |S|}$$

The maximum attacker accuracy on L is

$$\frac{1}{n_{col}(L)}$$

And in total, there cannot be more than $L_{max} + L_{seg}$ collision sets (one for each possible defended sequence length). Thus the MAA satisfies

$$\begin{aligned} a^M(S) &\leq (L_{max} + L_{seg}) \cdot |n_{col}(L)| \cdot \frac{1}{L_{seg} \cdot |S|} \cdot \frac{1}{n_{col}(L)} \\ &= \frac{L_{max} + L_{seg}}{L_{seg} \cdot |S|} \end{aligned}$$

□

Lemma 6 tells us that arithmetic sequence rounding sets and uniform distributions are very similar in the MAA. One can verify that, on average, the bandwidth overhead of the two schemes are equal as well. In this sense, we can consider it a formalization of the phenomenon we observed in Example 5. Lemma 4 shows that arithmetic sequence rounding sets are, on average, optimal over all deterministic sequence padding; and Lemma 5 shows that uniform distributions

reveal the least information over all random sequence padding. Optimizing over both deterministic and random sequence padding brings us to equal results.

The worst-case scenario for the arithmetic sequence rounding set occurs when there exists at least one packet sequence $P \in S$ where $|P| \in [mL_{seg} + 1, (m+1)L_{seg}]$ for every m (up to L_{max}/m), whereas that for the uniform distribution occurs when for every packet sequence $P \in S$ there exists at least one packet sequence P' with $P \leq P' \leq P + L_{seg} + 1$. In fact a data set that satisfies the worst-case scenario for the random distribution must also satisfy the worst-case scenario for the rounding set, but not vice-versa, so there is some possibility of the random distribution performing better. This point is somewhat minor in practice, as in either case the practical scenario is very close to the worst-case scenario.

5.2 Description of Website Fingerprinting Defenses

Website fingerprinting defenses adopt a diverse variety of approaches to thwart attacks. 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. We refer the reader to Section 5.1.5.2 for a strict definition of limited and general defenses. The code for all defenses is available for download with links in Appendix A.

5.2.1 Limited Defenses

As described in Section 5.1.5.2, limited defenses are designed specifically to defeat one or several known WF attacks. We implemented all defenses in this section and evaluate them in detail in Section 5.5.2, showing that they cannot hold up to `Wa-kNN`.

5.2.1.1 Adaptive padding (Adaptive)

Shmatikov and Wang[5] (2006) published adaptive padding [SW06] (hereafter referred to as Adaptive) as a defense against their own attack, `Sh-Timing`. Adaptive adds packets to cover interpacket times from the attacker.

Adaptive operates on bins of interpacket timing. Each bin b_i covers interpacket times between 2^{i-1} to 2^i milliseconds, with the exception that the smallest bin b_1 covers 0 to 2 ms. Adaptive

[5] Ming-Hsiu Wang.

chooses a target web page C, and for each bin the attacker learns the probability of an interpacket time from some packet sequence in C being in that bin.

To defend a packet sequence P, Adaptive attempts to pad P in such a way as to mimic C. For each packet (t, ℓ) in P (starting from the first), Adaptive picks a bin at random according to its occurrence probability in C. Let us suppose Adaptive picked bin b_i. If 2^i ms is larger than t, then Adaptive does not modify (t, ℓ) in any way and sends it normally. If 2^i ms is smaller than t_i, Adaptive sends a dummy packet $(2^i, \ell_{max})$ where ℓ_{max} is the MTU.

Shmatikov and Wang show that their defense is effective against `Sh-Timing`. We have seen that `Sh-Timing` relies heavily on interpacket timing and Adaptive chiefly attempts to cover interpacket timing.

5.2.1.2 Morphing packet length distributions (Morphing)

Wright et al. (2009) published traffic morphing [WCM09] (hereafter referred to as Morphing), a defense that randomly pads unique packet lengths so that these packet lengths look as if they came from a packet length distribution corresponding to another web page. They designed the defense for two purposes: VoIP classification and website fingerprinting. Morphing is to unique packet lengths what Adaptive is to interpacket times.

Morphing only changes the unique packet lengths of the packet sequence, and it does not act to cover sequence length, packet order, or interpacket timing. It is therefore especially effective against attacks that rely on unique packet lengths, such at `Li-Jac` and `He-MNBayes`.

With some web page C, the defense learns the probability distribution of packet lengths when accessing the page C, which we write here as C_L. $Pr(C_L = \ell)$ is the observed probability that any given packet when accessing C has length ℓ. Suppose we want to defend a packet sequence $P \notin C$, so that it looks like it comes from web page C. We morph each packet length $\ell_P \in P$ by drawing a random ℓ_C from the distribution of C_L. If $\ell_C \geq \ell_P$, then we pad ℓ_P to ℓ_C (with packet padding as in Section 5.1.5.2). Else, we send a packet of size ℓ_C, and we draw another ℓ'_C to attempt to cover the remaining packet of size $\ell_P - \ell_C$. We repeat the above procedure until the packet length we draw is enough to cover the remainder. The effect of this procedure is that the probability distribution of packet lengths in $D(P)$ will be the same as that of C_L.

Morphing optimizes the above process to minimize the bandwidth overhead. From C_L, Wright et al. derive a distribution $C_L^{\ell_P}$; the distribution depends on ℓ_P, the packet length we want to morph. Instead of drawing from C_L, they draw from $C_L^{\ell_P}$. $C_L^{\ell_P}$ is constructed such that, probabilistically, after morphing all packets in P, the distribution of packets in $D(P)$ will be the same as C_L as well. Note that this requires the participants of the defense to have trained on the web page that the client is about to visit.

Intuitively, the use of $C_L^{\ell_P}$ allows larger packets to morph into larger packets and smaller packets to morph into smaller packets, thus reducing overhead. They show that they can perform the above optimization with convex optimization. However, this creates a linkage between observed packet sequence lengths and true packet sequence lengths. In the worst case, Morphing using this optimization procedure may be reversible by a clever attacker. Cai et al. found that `Ca-OSAD` is unaffected by Morphing [CZJJ12], and we observed the same for `Wa-kNN` [WCN+14].

5.2.1.3 Obfuscation using HTTP methods (HTTPOS)

Luo et al. (2011) published HTTPOS (HTTP Obfuscation) [LZC+11]. They implemented the defense on the client side using specific features in HTTP. Unlike all other defenses we survey, HTTPOS does not require any support from a cooperator. It does so by cleverly using TCP advertised windows, HTTP pipelining, and HTTP ranges in order to control the sizes of both outgoing and incoming packets.

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

The authors also described several other possible defenses, such as manipulating interpacket timing and sending dummy packets, but the implementation of these are not described. We acknowledge that these defenses are possible within their platform; we implement and analyze only HTTPOS module 2.

Luo et al. have shown that this is a successful defense against older attacks including `Bi-Intertiming` and `Li-NBayes`. Cai et al. were the first to show success against HTTPOS with `Ca-OSAD`.

5.2.1.4 Loading decoy pages with real pages (Decoy)

Panchenko et al. [PNZE11] proposed a simple defense using background noise to defeat their own attack (`Pa-FeaturesSVM`). Under Decoy, whenever the client visits a page, she also loads a decoy page simultaneously. Therefore, the attacker cannot distinguish between the real and decoy pages.

The client cannot reveal her sensitive page accesses to Decoy. Therefore, the implementation of Decoy cannot use the monitored set as decoy pages. This means that it is still possible for an attacker to achieve a high accuracy under Decoy. As the decoy page comes from the open world, the attacker may not be interested in identifying the decoy page at all; he only wants to monitor a set of specific pages. Thus, we classify Decoy as a limited defense.

5.2.1.5 Tor's HTTP Pipelining defense (Pipeline)

Tor has implemented a WF defense [Per11a] in response to `Pa-FeaturesSVM`. Tor's defense uses HTTP pipelining (described in Section 2.1.1) by randomizing the depth of the pipeline (the maximum number of requests a connection can tolerate at once). Therefore, the order of requests may change if the number of requests exceeds the depth of the pipeline. Thus, we call this defense Pipeline.

Pipeline has no bandwidth overhead as HTTP pipelining does not introduce extra packets. Tor has updated Pipeline [Per13] recently in response to newer attacks. All Tor clients, by default, are using Pipeline: therefore, our Tor data set already included this defense. Therefore, we had shown that it is ineffective in Section 3.4. When we test other defenses in this section on the Tor data set, we would automatically be applying them on top of Pipeline.

5.2.2 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.

In response to this observation, we develop *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. We designed Tamaraw, Supersequence, and Walkie-Talkie, which are general defenses with different implementations of packet padding, packet scheduling, and sequence padding.

5.2.2.1 Buffered Fixed-Length Obfuscator (BuFLO)

Dyer et al. presented BuFLO [DCRS12] in 2012. BuFLO is the first defense that attempts to be general; Dyer et al. describe it as follows:

It is a realization of the "fool-proof" folklore countermeasure that, intuitively, should defeat any [WF] *classifier.*

Dyer et al. test BuFLO under several combinations of packet scheduling and sequence padding parameters. In this work, we only test BuFLO under the parameters that minimize the MAA, because other parameters give BuFLO an MAA very close to 1, which undermines BuFLO as a general defense. The reader may refer to the original work by Dyer et al. for other variations.

Packet padding. All packets are padded to 1500 bytes.

$$\begin{aligned} \ell_{ppad+} &= 1500 \\ \ell_{ppad-} &= 1500 \end{aligned}$$

Packet scheduling. Packet scheduling proceeds at a regular value ρ:

$$\begin{aligned} Z_{sched+} &= \{0, \rho, 2\rho, 3\rho, \ldots\} \\ Z_{sched-} &= \{0, \rho, 2\rho, 3\rho, \ldots\} \end{aligned}$$

$\rho = 0.02$ s in BuFLO.[6]

Sequence padding. Sequence padding only occurs if the total number of packets was under some number L_{max}.

$$\begin{aligned} Z_{spad+} &= \{L_{max}\} \\ Z_{spad-} &= \{L_{max}\} \end{aligned}$$

$L_{max} = 500$[7] in BuFLO, corresponding to 10 seconds of data.

If the total number of packets is above L_{max}, BuFLO performs no sequence padding. For their implementation, Dyer et al. tested L_{max} up to a maximum value corresponding to 10 s of packet delivery. They found that BuFLO still failed often against WF attacks. This is because on Tor, page load time often exceeds 10 s, especially since BuFLO slows down page loading significantly, due to its rate-limited packet scheduling. BuFLO is still a general defense, as all packet sequences with $|P_+|, |P_-| \leq L_{max}$ will collide, but such packet sequences are in the

[6] When implementing BuFLO for a Tor cell sequence, we set $\rho = 0.0067$ s so that there are three times as many cells but roughly the same amount of data per second. This is because each cell in Tor is about one-third the size of the MTU.

[7] As with packet scheduling, on Tor we set L_{max} = 1500.

minority in our data sets. As BuFLO is not our work, we do not attempt to further optimize its choice of parameters L_{max} and ρ.

The chief weakness of BuFLO is that the choice of L_{max} is awkward. Increasing L_{max} increases the number of web pages BuFLO collides, but it also increases the bandwidth and time overhead. There is no well-established method to select L_{max}. We solve this problem with Tamaraw, as below.

5.2.2.2 Effective and efficient BuFLO (Tamaraw)

To improve on BuFLO, we developed a new defense, Tamaraw (2014).[8] Tamaraw can be viewed as a version of BuFLO taking a different approach in each of the three defense components. Tamaraw boasts significantly greater efficiency than BuFLO.

Packet padding. In BuFLO, all packets were padded to the MTU, which was 1500 bytes in our network. In Tamaraw, we instead pad packets to some ℓ_{ppad+} and ℓ_{ppad-}. If the size of an outgoing packet is greater than ℓ_{ppad+}, we split the packet there, and similarly for incoming packets. The intention of selecting ℓ_{ppad+} and ℓ_{ppad-} instead of simply 1500 is to reduce the bandwidth overhead, because many packets have a packet length much less than 1500.

Splitting packets itself carries a bandwidth overhead, as we will need more packet headers; it also carries a time overhead, as packet scheduling delays the split packet. In Section 5.5.3, we will evaluate the effect of ℓ_{ppad+} and ℓ_{ppad-} on packet padding. We will find that $\ell_{ppad-} = 1500$ is in fact optimal for incoming packets, but not for outgoing packets.

Packet scheduling. For packet scheduling, we treat outgoing and incoming packets differently, with two rates ρ_+ and ρ_-:

$$\begin{aligned} Z_{sched+} &= \{\rho_+, 2\rho_+, 3\rho_+, \ldots\} \\ Z_{sched-} &= \{\rho_-, 2\rho_-, 3\rho_-, \ldots\} \end{aligned}$$

Compared to BuFLO, where $\rho_+ = \rho_-$, this is another way for Tamaraw to achieve a lower bandwidth and time overhead. Our choice of ρ_+ will be much greater than ρ_-, simply because we expect many more incoming packets than outgoing packets: on our data set, the number of incoming packets is about 10 times that of the number of outgoing packets. Experimentally, we determine the optimal parameters of ρ_+ and ρ_- in Section 5.5.3.

[8] A tamaraw is a smaller type of buffalo.

Sequence padding. Rather than rounding all sequence lengths to a single L_{max} in BuFLO, we instead select the rounding sets Z_{spad+} and Z_{spad-} with some parameter L_{seg}:

$$\begin{aligned} Z_{spad+} &= \{L_{seg}, 2L_{seg}, 3L_{seg}, \ldots\} \\ Z_{spad-} &= \{L_{seg}, 2L_{seg}, 3L_{seg}, \ldots\} \end{aligned}$$

Z_{spad+} and Z_{spad-} operate independently of each other; $|D(P)_+|$ and $|D(P)_-|$ can be different multiples of L_{seg}.

We can see that Tamaraw is a general defense. After applying packet delivery and packet padding, the only difference between two packet sequences is the total number of incoming and outgoing packets. These are different multiples of L_{seg}. If L_{seg} is large enough, then there is a significant chance that any two given packet sequences will have the same sequence lengths.

The choice of parameter L_{seg} affects the MAA, bandwidth overhead, and time overhead of the scheme. A larger L_{seg} increases the collision rate as there are fewer possible multiples of L_{seg} within the range of most packet sequences, but it also increases the overhead as sequence padding requires up to L_{seg} packets.

On the other hand, ρ_-, and ρ_+ affect the time/bandwidth overhead tradeoff of the scheme. A larger ρ_- and ρ_+ increases time overhead by forcing packets to wait longer for the correct timing window dictated by Z_{sched+} and Z_{sched-}; it also decreases bandwidth overhead because it is less likely that there was no real content to send in the intervals ρ_- and ρ_+.

In Section 5.5.3, we will investigate the optimal choice of parameters L_{seg}, ρ_- and ρ_+ from a training data set, and their effect on the tradeoff between a^M, O^B and O^T.

5.2.2.3 Smallest common supersequence (Supersequence)

We presented Supersequence (2014) as a defense against `Wa-kNN` in the same work [WCN+14]. Supersequence attempts to find the bandwidth-optimal simulatable defense. Supersequence is based on our observation in Section 5.1.6: the bandwidth-optimal simulatable, deterministic defense D on a collision set is the shortest common supersequence (SCS) of its elements. Finding the SCS is an NP-hard problem, so we will only be able to approximate it.

We can think of Tamaraw and Supersequence as having opposite approaches: Tamaraw starts with the most inefficient defense (sending packets constantly all the time) and describes procedures to improve efficiency, whereas Supersequence starts with the most efficient defense (SCS) and attempts to approximate such a defense.

We describe Supersequence in greater detail in Section 5.3; here, we only describe its general defense components briefly to enable comparison with other defenses.

Packet padding. We did not discuss packet padding of Supersequence in the original paper because we only tested it on Tor cell sequences. In our experiments, we will use the same method as Tamaraw, with separate incoming and outgoing packet lengths.

Packet scheduling. Supersequence learns Z_{sched} by finding the SCS of collision sets. As supersequence learning depends significantly on our assumptions of the participants' capacity, we describe this process in detail in Section 5.3.

Sequence padding. Supersequence uses the same sequence padding mechanism as Tamaraw: packet sequences are padded to some multiple of L_{seg}. However, we will see that Supersequence packet scheduling leaves only the total sequence length as a feature for attacks, rather than both the outgoing and incoming sequence lengths. While Tamaraw pads outgoing and incoming sequence length to (possibly) separate multiples of L_{seg}, Supersequence sequence padding pads the total sequence length to one multiple of L_{seg}.

5.2.2.4 Half-duplex communication (Walkie-Talkie)

Previous general defenses BuFLO, Tamaraw and Supersequence all have a significant problem: the time overhead O^T is very large. Depending on the parameters of the defense, the time overhead ranges between 100% and 400%. This is prohibitively slow, especially on low-latency anonymous networks like Tor, for which it already takes more than 10 seconds to load each page on average. Packet scheduling D_{sched} causes such a large time overhead, as it schedules many packets to wait for the next timing window. Each packet that is forced to wait delays the entire packet sequence, and the total time to send the packet sequence adds up to a much larger number.

We solve this problem with Walkie-Talkie, a defense that has a highly tunable bandwidth overhead and almost no time overhead, and yet meets the criteria of being a general defense. Walkie-Talkie has two components:

1. Half-duplex communication. Half-duplex communication refers to the way walkie-talkies work: only one side of the conversation can be speaking at a time. 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 Walkie-Talkie, 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 use half-duplex communication because it significantly reduces the

feature set available to the website fingerprinting attacker, thus decreasing his accuracy: only the number of packets in each burst is available.

2. Random sequence padding. We randomly select a number of packets to add to each burst, including fake bursts. This collides packet sequences in the manner described in Section 5.1.5.3.

We describe the implementation of half-duplex communication in the browser in Section 5.4. Here, we describe the use of general defense components in Walkie-Talkie to compare it with previous general defenses.

Packet padding. Since we have only implemented Walkie-Talkie on Tor Browser, we will not test it on non-Tor packet sequences. Tor already has constant-size cells, so we do not need to add any packet padding.

Packet scheduling. Walkie-Talkie uses a different packet scheduling mechanism compared to all other known general defenses. Other general defenses use what we refer to as *fixed-rate packet scheduling*, where a scheduling function Z_{sched} determines exactly when packets can and must be sent. Walkie-Talkie has no such function, which may seem surprising considering its claim to be a general defense. Instead, Walkie-Talkie uses half-duplex communication for packet scheduling. Each participant only sends packets immediately when she has accumulated the maximum possible amount of real information. With previous general defenses, fixed-rate packet scheduling increases time and bandwidth overhead significantly because each side must either delay packets or send fake packets. We will discuss this significant point in greater detail in Section 5.4.

Sequence padding. Unlike previous defenses, Walkie-Talkie employs random padding. We discussed random padding in Section 5.1.5.3. Walkie-Talkie pads the sequence in two ways:

1. *Padding real bursts.* Bursts of real packets reveal the length of each burst. We add a random number of outgoing and incoming packets to cover this number. When the attacker counts the number of packets in the defended packet sequence, he must take into consideration the fact that the real number of packets is smaller, limited by the maximum number of fake packets added. We will evaluate several distributions of random numbers.

2. *Adding fake bursts.* The total number of bursts is also an informative feature for the attacker. Furthermore, the position of large and small bursts (since burst padding only has

Table 5.1: Categorization of all defenses we implement and investigate. Defenses can be limited or general; simulatable or non-simulatable; and deterministic or random.

Defense	Limited/ General	Simulatable/ Non-Simulatable	Deterministic/ Random
Adaptive [SW06]	Limited	Simulatable	Random
Morphing [WCM09]	Limited	Simulatable	Random
HTTPOS [LZC+11]	Limited	Non-Simulatable	Random
Decoy [PNZE11]	Limited	Simulatable	Random
Pipeline [Per11a]	Limited	Non-Simulatable	Random
BuFLO [DCRS12]	General	Simulatable	Deterministic
Tamaraw [CNW+14]	General	Simulatable	Deterministic
Supersequence [WCN+14]	General	Simulatable	Deterministic
Walkie-Talkie [WG15b]	General	Non-Simulatable	Random

a limited ability to change its size) is also informative. At random between real bursts, we add bursts of packets that seek to look like real bursts. This also serves to change the position of large and small bursts.

Our analysis of Walkie-Talkie as a general defense follows the same definition of the perfect attacker as in Section 5.1.5.2. We also found that though it is a general defense, Walkie-Talkie also defeats known attacks with very little overhead. We analyze Walkie-Talkie in detail in Section 5.4.

5.2.3 Categorization

We can classify defenses into several categories using the terminology we developed in Section 5.1. A defense is general if it collides packet sequences and limited if it cannot. We can also classify defenses as simulatable or non-simulatable, and deterministic or random, based on the definitions in Section 5.1.5.1 and Section 5.1.5.3. In Table 5.1, we list all defenses we investigate, and we give a description of our categorization as follows:

Non-simulatable, random: This includes Pipeline, HTTPOS, and Walkie-Talkie, our newest website fingerprinting defense. All three defenses change the browser. Pipeline enables HTTP pipelining for the browser with random pipeline depths. HTTPOS modifies the client's browser, allowing the client to hide unique packet lengths by sending an HTTP

range request strategically. Walkie-Talkie uses half-duplex communication by stopping the client from sending requests until all responses are complete. Pipeline, HTTPOS and Walkie-Talkie all require changes to browser code. These defenses are random because Pipeline changes the browser behaviour in a random manner, HTTPOS splits packets randomly, while Walkie-Talkie uses random sequence padding.

Non-simulatable, deterministic: We can also implement Walkie-Talkie sequence padding so that it is deterministic. Thus, the overall defense will be non-simulatable and deterministic. However, we will see that this variant of Walkie-Talkie is not efficient.

Simulatable, random: This includes Adaptive, Morphing and Decoy. Adaptive changes interpacket timing distributions of a defended packet sequence to that of a packet sequence from a second packet sequence by drawing interpacket times from the second packet sequence at random, while Morphing does so for packet lengths. Decoy loads a random decoy page with each real page.

Simulatable, deterministic: This includes BuFLO, Tamaraw, and Supersequence. All of these defenses use fixed-rate packet scheduling and deterministic sequence padding. Both Tamaraw and Supersequence attempt to minimize bandwidth overhead, but defenses in this class often have a very high time overhead.

5.3 Supersequence

In Section 5.1.6, we saw that the bandwidth-optimal simulatable packet scheduling procedure involves finding the supersequence of collision sets. We created Supersequence based on this observation. In this section, we describe Supersequence in detail, expanding on the brief description in Section 5.2.2.3.

Denoting $f_{scs}(S)$ as the shortest common supersequence of all packet sequences in the set S, we want to find f_{scs}. However, finding such an optimal solution requires solving two hard problems:

Collision set selection. We can achieve bandwidth optimality by applying f_{scs} on a specified collision set, but this does not immediately give us bandwidth optimality in the entire data set S. For example, if the collision set contains packet sequences of similar length, it is likely to have a lower bandwidth overhead than otherwise. Given the set of all possible packet sequences, we want to group them into collision sets to minimize the overhead, for a given bound on attacker accuracy. Choosing the correct collision sets is difficult, and we discuss this in Section 5.3.1.

SCS as an NP-hard problem. Finding $f_{scs}(S)$ is in general NP-hard. [JL95] We discuss how we attempt to work around this issue in Section 5.3.2.

5.3.1 Collision Set Selection

To achieve bandwidth optimality for Supersequence, we want to select the right collision sets. The client has restricted information: a realistic client must decide on which supersequence to use (which collision set to choose) before seeing the packet sequence. While the client can gain information that assists her in making this decision (the URL, previous page load data, training data, information about the client network, the first few packets of the sequence, etc.), the mere storage and usage of this information carries additional privacy risks. In particular, Tor Browser keeps no disk storage (including no cache except from memory), so that storing extraneous information puts the client at additional risk. In this section, we describe how realistic conditions impose restrictions on the power of the client to choose collision sets.

We formalize this observation by imposing additional limits on anonymity set selection in the defense D trained on S. We define three levels of information for a client applying a simulatable, deterministic website fingerprinting defense:

No information. The client has no information about the packet sequence to be loaded. All sequences map to a single collision set. This level of information is far too expensive in both bandwidth and time overhead because all packet sequences are padded to the longest packet sequence that the participants want to defend, and packet sequences differ significantly in length. We will not investigate this level of information further because it is not practical.

Sequence end information. The client knows when the sequence has ended, but this is the only information the client gets about the packet sequence. We phrase this level of information as a restriction on D. D only outputs prefixes (of various lengths) of a fixed single common supersequence: for any P, P', such that $|D(P)| \geq |D(P')|$, the first $|D(P')|$ packets of $D(P)$ are exactly $D(P')$. We say that $D(P')$ is a prefix of $D(P)$. With sequence end information, for any P and P' in S, either $D(P)$ is a prefix of $D(P')$ or $D(P')$ is a prefix of $D(P)$.

Class-level information. The client knows the URL of the page. To learn about the page, the client has loaded the page before and performed some form of offline training to learn about the page. The client cannot distinguish between different packet sequences of the same page (even though the page may be multi-modal). so defended packet sequences of the

Table 5.2: Relationship between three levels of information and the restrictions on $D(P)$ and $D(P')$ for some defense D under this level of information and two packet sequences $P, P' \in S$.

Level of Information	Restriction on $D(P)$ and $D(P')$
No information	$\forall P, P' \in S, D(P) = D(P')$
Sequence end information	$\forall P, P' \in S$, $D(P)$ is a prefix of $D(P')$ or vice-versa
Class-level information	For any class C of packet sequences in S from the same page, $\forall P, P' \in C$, $D(P)$ is a prefix of $D(P')$ or vice-versa

same page must be prefixes or suffixes of each other (as with sequence end information). Defended packet sequences of different pages do not have any such restriction.

We give a brief mathematical definition of the three levels of information in Table 5.2.

Optimality under the above levels of information requires the computation of supersequences over collision sets. Each level of information places different restrictions on what collision sets are allowed. We describe our methodology to find collision sets under each level of information:

No information. All packet sequences are sent under the same supersequence, so we do not have to choose collision sets.

Sequence end information. As above, there is only one supersequence. What is different is that possible outputs of the defense correspond to a prefix of the one supersequence, terminating according to some rounding set Z_{spad}. This is similar to sequence padding in Tamaraw. We can also choose collision sets such that the MAA for each packet sequence is less than some parameter a_{max}. We will evaluate those schemes in Section 5.5.4.

Class-level information. With class-level information, the defense participants know the true web page of the packet sequence being transmitted. The client sees the true web page, and the client can tell the cooperator.

Compared to sequence end information, where all packets are scheduled under a single supersequence, with class-level information we are permitted to schedule packet sequences from different web pages under different supersequences. This gives us an extra degree of freedom: we can choose which web pages should have the same supersequence. However, it also complicates the issue of finding optimal collision sets. Packets scheduled under different supersequences belong to different collision sets, so we must group packet sequences carefully. Class-level information does not change our sequence padding strategy: we still

use rounding sets containing multiples of some L_{seg}, much like Supersequence with sequence end information and Tamaraw.

For class-level information, we schedule packets using *clustering*. Clustering is a class of unsupervised machine learning techniques (unlike WF attacks in Chapter 3, which use supervised machine learning techniques). First, we find $f_{scs}(C)$ for all C, the supersequence of all elements in each class, which we refer to as class sequences. Then, we cluster class sequences into n_{seq} classes to decide which supersequence the client should use when loading a page within each class.

We use a variation of k-means clustering to find clusters. k-means clustering works as follows. Out of all class sequences, we randomly pick k and label them as centers and other class sequences as non-centers. We assign each non-center to the cluster with the closest center. Then, we re-compute cluster centers as the mean of each cluster. Since the cluster centers have changed, we re-assign each non-center to the cluster with the closest center again. We re-iterate the above until convergence.

k-means clustering thus requires an informative distance (like k-NN) and a way to compute the mean. We choose the distance for two class sequences P and Q is as follows. Given P and Q, we write P', Q' as the first $\min\{|P|, |Q|\}$ packets of P and Q respectively. Then we write the distance $d_{scs}(P, Q)$ as

$$d_{scs}(P, Q) = |f_{scs}(P', Q')|/|P'| - 1$$

This distance is exactly the overhead O^B of padding P' to $f_{scs}(P', Q')$. It is smaller if P' and Q' are more similar. We do not consider the whole of P and Q because the sequence length will be tackled by sequence padding. For example, if Q is a prefix of P, scheduling Q under P would result in zero overhead, so we want the clusters to reflect this phenomenon and minimize overhead. We choose the mean of each cluster as the point that has the minimum total distance with all other points in the cluster.

Note that clustering increases the number of collision sets. Supposing that L_{max} is the length of the longest packet sequence after packet scheduling, the maximum number of collision sets in sequence end information is L_{max}/L_{seg}, but it will be kL_{max}/L_{seg} with class-level information, where k is the number of clusters. We adjust for this by multiplying L_{seg} by k under class-level information so we can have the same MAA. We evaluate class-level information in Section 5.5.4.

Interestingly, we can bound the packet scheduling bandwidth overhead of Supersequence on any data set. Note that the bandwidth overhead of packet and sequence padding are relatively minuscule (around 5–10%) compared to packet scheduling overhead.

Lemma 7. On any data set S, Supersequence can achieve a packet scheduling bandwidth overhead under 100%.[9]

Proof. Consider the packet scheduling Z_{sched+}, Z_{sched-} as follows:

$$\begin{aligned} Z_{sched+} &= \langle T, 2T, \ldots \rangle \\ Z_{sched-} &= \langle T, 2T, \ldots \rangle \end{aligned}$$

Suppose we choose T to be large enough that it is greater than any interpacket timing $t_i \in P_t$ for $P \in S$. T is the same for Z_{sched+} and Z_{sched-}.

Consider sending any sequence $P \in S$ under this packet scheduling. Suppose the next real packet to send is $P_i = (t_i, \ell_i)$, and the time is currently nT. At time $(n+1)T$ we will send an outgoing packet and an incoming packet according to Z_{sched+} and Z_{sched-}. Since $T > t_i$, if $\ell_i = -1$, then $Z_{sched-n+1}$ will be a real packet with the contents of p_i; if $\ell_i = 1$, then $Z_{sched+n+1}$ will be the real packet.

Therefore, only at most one of the two packets dictated by Z_{sched+} and Z_{sched-} will be a fake packet; the other must be real. Repeating this process over the entire sequence P, at most half of all packets we send under packet scheduling will be fake. □

Lemma 7 applies to Tamaraw as well. One implication of the above is that Tamaraw and Supersequence must be cheaper than Decoy in bandwidth overhead. It should be said, however, that a 100% bandwidth overhead is not a small number. Further, the proof allows for a very large time overhead. In our evaluation of Supersequence, we will find that we can do somewhat better than a 100% bandwidth overhead given a data set.

5.3.2 SCS as an NP-hard Problem

For packet scheduling, supersequence uses the shortest common supersequence (SCS) of collision sets found in Section 5.3.1. We can find the SCS of two packet sequences using dynamic programming in $O(n^2)$ time, where n is the length of the two packet sequences. However, the SCS of multiple sequences is in general NP-hard [JL95].

Our problem specifically is a fixed-alphabet SCS problem, since our sequences only have two possible letters after packet padding (outgoing packet length ℓ_{ppad+} and incoming packet length ℓ_{ppad-}), but this is still NP-hard. The best known algorithms for finding the fixed-alphabet SCS of k sequences of length n perform at an algorithmic complexity of $O(n^k)$; in practical

[9] Note that this is true for any level of information.

implementations, the run time may become overwhelming with $k > 6$ [Pie03]. $k = 6$ may not be sufficiently high for an effective general defense.

To tackle the NP-hard problem, we use a simple algorithm for approximating:

$$f_{scs}(\{P^1, P^2, \ldots, P^n\})$$

This algorithm can be thought of as a repeated voting scheme to decide the next packet of the supersequence, where each input sequence is a voter. We start with an empty supersequence, and define a counter for each packet sequence denoted as $c_1, c_2, \ldots, c_n$, where each counter starts at 1, i.e. $c_1 = c_2 = \ldots = c_n = 1$.

For each vote, we count the number of input sequences for which the c_i-th element of P^i is an outgoing packet. If the number exceeds $n/4.5$, we append an outgoing packet to the supersequence, and increment all c_i for which the c_i-th element of P^i is an outgoing packet by 1. Else, we append an incoming packet, and increase the corresponding counts by 1. We do this iteratively until each counter c_i becomes $|P^i| + 1$, and the algorithm terminates. The choice of $n/4.5$ is because for web page loading, there are fewer outgoing packets than incoming packets, and this choice reduces our overhead significantly.

We note that it is easy to construct cases where the above algorithm performs very poorly. In fact, it is known that any polynomial-time approximation algorithm of shortest common supersequences cannot have bounded error [JL95]. We refer the reader to Turner for a review of SCS approximation algorithms [Tur89].

Time overhead. The above algorithm does not define the timing of Z_{sched}; rather, it only defines a packet order of incoming and outgoing packets. We presented the above algorithm in our original work on Supersequence [WCN+14], where we had optimistically disregarded the time overhead by assuming that the supersequence packet times were large enough to always send packets of the right direction. This is the same assumption we made for Lemma 7.

In this work, we augment the above algorithm to define Z_{sched+} and Z_{sched-} fully. In addition to a counter, each packet sequence also has a queue time $t_q^1, t_q^2, \ldots, t_q^n$. The queue time of each packet sequence indicates how long we must wait until the next packet in the sequence can be sent; it starts at 0. After selecting which direction the next packet should be sent in—i.e., whether we are going to append to Z_{sched+} or Z_{sched-}—we examine the queue time of all packet sequences where the next packet is in the same direction. Then, we select the a time T such that at least a fraction p_{vote} of all observed queue times are under T. Next, we decrease all queue times by T, to a minimum of 0. We only increment the counter c_i for any packet sequence P^i where the packet is in the right direction *and* $t_q^i = 0$. When a counter is incremented for some

P^i, and the next packet is in the opposite direction of the previously sent packet, then we set the queue time t_q^i to be the interpacket time between those packets. We pessimistically assume that adjacent packets in opposite directions are logically dependent on each other (a HTTP request triggering a response and vice-versa), and the interpacket time could be due to network latency, for which we must wait. p_{vote} is thus a parameter of Supersequence, between 0 and 1, that trades off bandwidth overhead for time overhead.

5.4 Walkie-Talkie

In this section, we describe Walkie-Talkie, the only published general defense that does not use fixed-rate packet scheduling. As discussed in Section 5.2, BuFLO, Tamaraw and Supersequence use fixed-rate packet scheduling, which has several problems:

Overhead. A major barrier towards the implementation of general defenses is that the bandwidth overhead and time overhead are both very large. Under fixed-rate packet scheduling, both the client and the cooperator send many dummy packets. We will find that both the bandwidth and time overhead of Supersequence and Tamaraw are around 100% to 200%, depending on the desired MAA, in Section 5.5. For Tor, the time overhead implies that loading a single web page will take more than half a minute. These defenses may be overkill for some clients.

Implementation. Fixed-rate packet scheduling leads to an inflexible packet rate. Cai et al. [CNJ14] have found that the fixed packet rate demanded by Tamaraw and Supersequence causes poor behaviour in congested networks, especially when other protocols are also demanding bandwidth, since there is no congestion control in fixed-rate packet scheduling. 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 for packet scheduling, Walkie-Talkie addresses both of these problems. Its advantages are as follows:

Overhead. Fixed-rate packet scheduling has a high time overhead because it requires 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. Furthermore, half-duplex communication has no bandwidth overhead; the only bandwidth overhead in our defense, Walkie-Talkie, comes from sequence padding, described later in Section 5.4.5.

Implementation. Half-duplex communication is easy to implement; we have already done so, and we provide details on our implementation in Section 5.4.2.

Walkie-Talkie is thus a general defense with low overhead and no restriction on packet rate. We continue by discussing the design restrictions of Walkie-Talkie.

5.4.1 Design Goals

We designed Walkie-Talkie with the following goals in mind:

1. General: Walkie-Talkie seeks to be a general defense; even a perfect attacker would not be able to identify a page because of packet sequence collisions.
2. Easy to use: The client does not need to configure Walkie-Talkie. 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. Some defenses, such as Adaptive, Morphing, HTTPOS and Supersequence (class-level information), require the client to train on some packet sequences; Walkie-Talkie does not.
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.

Next, we describe how we implement half-duplex communication.

5.4.2 Implementing Half-duplex Communication

We implement half-duplex communication by changing the way the client's browser works. The destination server does not need to make any changes to tolerate Walkie-Talkie; the client simply does not talk when the destination server is talking.

In Section 2.1.1 we gave a basic overview of how browsers work. We described how browsers use persistent connections to load data from a web server with transactions. A pending transaction queue keeps track of transactions that cannot be dispatched immediately.

Within Firefox, the code that handles all of the above is known as a connection manager. 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.

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.

5.4.3 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[10] 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. [Per11b] 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.

5.4.4 Other Implementation Details

We make several further changes to the Tor Browser to ensure half-duplex communication:

[10]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.

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 it is currently disabled in major browsers because of misbehaving servers [Moz14, Goo]; many web servers do not support pipelining [LZC+11]. Tor Browser has enabled pipelining as a costless defense against WF [Per11a], but we have seen that it has no noticeable effect on `Wa-kNN`. 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 premaster 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 under TLS a short while after a burst has started. We have not yet implemented the TLS subsystem for `Walkie-Talkie`; in our experiments, we assume that this subsystem has been implemented and all packets occur at the start of the burst.

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.

Long-lived resources. Some web pages have long-lived resources, such as video streams or interactive elements such as chat. Rather than waiting for these elements to finish loading before loading any further resources, which would hamper page load, we specify that resources that take more than 5 seconds to load will not be counted for half-duplex communication blocking. As a result, Walkie-Talkie will not attempt to defend these resources. This is acceptable, as it is difficult to defend the presence of such a resource in any case, as the bandwidth overhead required to do so for any defense would overwhelming. Walkie-Talkie will still attempt to defend the bursts of the underlying page as usual.

5.4.5 Padding

We start by assuming that the client has already applied packet padding, so that all packets have the same size (or we are using Tor cell sequences). 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*. Thus, we write a packet sequence

$$P_b = \langle b_1, b_2, \ldots \rangle$$

n_{burst} is the number of bursts in packet sequence P. Each $b_i = \langle b_{i+}, b_{i-} \rangle$ where b_{i+} is the number of outgoing packets in the burst and b_{i-} is the number of incoming packets. Note that in the packet sequence P, the outgoing packets precede the incoming packets in time. Also note that P_b contains no packet timing information; it is a shortened way of writing P_ℓ. The attacker's information is then limited to a list of pairs of positive integers.

The client adds dummy packets that contain no information in order to lower the perfect attacker's MAA. Recall that the perfect attacker can only fail when a *collision* occurs; that is, when the same packet sequence (with dummies) could have come from at least two different web pages. As with other general defenses, we want to minimize the MAA and the overhead.

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.

It is not necessary to add enough padding for the MAA to be close to zero, because of the base rate fallacy (see Section 2.2.2). The base rate fallacy applies to the perfect attacker as well. 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% [WCN+14, Per13] even if the attacker's true positive rate was 100%. The number of false alarms will overwhelm the attacker's classifier.

5.4.5.1 Deterministic padding

For deterministic padding, we perform *rounding* on burst sizes as described in Section 5.1.5.3.

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 have a different rounding set for each burst, and we use Y_{i+} to denote padding the i-th outgoing burst b_{i+}, and similarly for incoming bursts. This method of causing collisions is similar to Tamaraw; and much like Tamaraw, it is more efficient to have

distinct rounding sets for outgoing and incoming traffic. We sometimes elide the distinction between different rounding sets for simplicity of discussion in what follows. The overhead of the defense (the proportion of fake packets added) is therefore

$$O_D^B(P) = \frac{\sum_{i=1}^{|P_b|}(round_{Y_{i+}}(b_{i+}) + round_{Y_{i-}}(b_{i+}))}{\sum_{i=1}^{|P_b|}(b_{i+} + b_{i-})}$$

We use a deterministic algorithm to find the overhead-optimal rounding set Y given its size $|Y|$. Recall from Section 5.1.6 that the choice of rounding sets does not affect the MAA for the most conservative case when all pages produce consistent packet sequences, so we only have to optimize the overhead.

Suppose that the set of integers for which we want to find the optimal Y is a multi-set B. For example, B can be the set of first incoming burst sizes across all observed sequences. B is sorted from small burst sizes to large burst sizes. Denote $Y_{\leq}(a)$ as $\{y \in Y : y \leq a\}$ (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. Note that this is conceptually similar to the case of Supersequence, for which we cannot find a supersequence for a collision set greater than 6 (see Section 5.3.2); however, the computational limit of Supersequence induces a lower bound on the MAA we can achieve while the computational limit of Walkie-Talkie induces an upper bound. Hence we do not need to investigate an approximation algorithm to achieve a low MAA.

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.

5.4.5.2 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 $X : \mathbb{Z}_{\geq 0} \rightarrow [0, 1]$. Given a packet sequence $P_b = \langle b_1, b_2, b_3, ..., b_{|P_b|} \rangle$ with $b_i = (b_{i+}, b_{i-})$, we apply defense D as follows to produce $D(P_b)$:[11]

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

2. Adding fake burst pairs: From two distributions $X_{i+_{fake}}$ and $X_{i-_{fake}}$, we draw $x_{i+_{fake}}$ and $x_{i-_{fake}}$, and generate a new fake burst pair $(x_{i+_{fake}}, x_{i-_{fake}})$. We add fake burst pairs at random with probability p_{fake} 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+}, X_{i-}, $X_{i+_{fake}}$, $X_{i-_{fake}}$ (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 decrease the MAA by increasing overhead). In the following we describe how we choose these distributions.

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:

[11] 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.

- 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 burst 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.

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 $X_{+_{fake}}$ and the number of incoming packets is drawn from $X_{-_{fake}}$. 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 5.5.5 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.

5.5 Evaluation

Here, we evaluate all website fingerprinting defenses described in Section 5.2. We start by describing our data collection procedure in Section 5.5.1. First, we will show that limited defenses tend to fail against `Wa-kNN` in Section 5.5.2, thus motivating the use of general defenses. We evaluate Tamaraw, Supersequence and Walkie-Talkie separately in Section 5.5.3, Section 5.5.4, and Section 5.5.5. Each of these defenses is tunable, with parameters that allow tradeoffs between maximum attacker accuracy, bandwidth overhead, and time overhead: we investigate this and other phenomena for each defense. We will also evaluate alternative Walkie-Talkie constructions in Section 5.5.6. We next compare general defenses in Section 5.5.7 on the bases of maximum attacker accuracy, bandwidth overhead, and time overhead. We will find that Walkie-Talkie is suitable for low overheads, while Tamaraw and Supersequence are more powerful if the user can accept high overheads (over 100% bandwidth and time overhead).

5.5.1 Data Collection

To evaluate BuFLO, Tamaraw and Supersequence, we use the same Tor data set as in Section 2.4. Since they are simulatable defenses, we can simply apply them on packet sequences collected with the regular Tor Browser to compute their overhead and maximum attacker accuracy. On the other hand, Walkie-Talkie is not simulatable, so we need to collect a new data set.

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 5.4.2. We collected data from Alexa's top pages [Ale15], much like our Tor data set. 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 collected half-duplex communication traces for Tor (the non-simulatable part of the Walkie-Talkie defense), but we did not implement sequence padding (the simulatable part of the defense) using Tor. Rather, we simulated the sequence padding 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.

Table 5.3: List of limited defenses we implemented and how we configured them to compare them on the same basis. The second column indicates whether we use the Tor or non-Tor data set. We use the non-Tor data set when the defense is unsuitable for Tor.

Defense	Data set	Implementation
Adaptive	Tor	For all sites, we mimicked the interpacket times of an instance of google.com.
Morphing	Non-Tor	For all sites, we mimicked the packet lengths of an instance of google.com.
HTTPOS	Non-Tor	We implemented HTTPOS module 2 (splitting unique packet lengths randomly).
Decoy	Tor	We chose random non-monitored pages as the decoy; the attacker is not aware of those web pages.

Table 5.4: List of limited defenses and their bandwidth and time overhead as well as the accuracy of `Wa-kNN` on them in the closed-world 100×100 data set.

Defense	Bandwidth overhead	Time overhead	`Wa-kNN` accuracy
Adaptive	61 %	0 %	0.602 ± 0.005
Morphing	50 %	0 %	0.936 ± 0.002
HTTPOS	6.4 %	0 %	0.956 ± 0.002
Decoy	110 %	25 %	0.182 ± 0.004

5.5.2 Limited Defenses

In this section we test the effectiveness of limited defenses. We attack data sets guarded by Adaptive, Morphing, HTTPOS and Decoy using `Wa-kNN`. As these defenses work under slightly different assumptions, we describe how we implement them in Table 5.3 to compare them fairly. Details of each defense are given in Section 5.2. We do not separately evaluate Tor's Pipeline defense because it is already contained in the Tor data set. The reader can consider the results in Table 3.6 to be an evaluation of Pipeline.

We perform all experiments in this subsection in the closed-world 100×100 data set. The attacker is aware of the WF defense the client uses and applies it to his data set. We present the accuracy of `Wa-kNN` against each defense in Table 5.4.

From Table 5.4, we make the following observations about each limited defense:

Adaptive. Unexpectedly, Adaptive is quite expensive in terms of its bandwidth overhead. In

our implementation, we found that the smallest bin of interpacket times (0.002 s) had 88% of all packets; this is the probability of Adaptive selecting the 0.002 s bin. Recall that if Adaptive does not see real content in the maximum time allowed by the bin, it sends a fake packet. The large number of fake packets only decreased the `Wa-kNN` accuracy to 0.6, as the dummy packet locations contain significant information about the true packet sequence.

Morphing. Morphing has a comparable overhead to Adaptive, but it only barely dents the accuracy of `Wa-kNN`. This is because it only covers unique packet lengths, and `Wa-kNN` extracts many other features.

HTTPOS. HTTPOS has almost no overhead since it only splits packet lengths. The small overhead comes from padding outgoing packets to size 1500. For similar reasons as Morphing, it has no effect on `Wa-kNN`.

Decoy. This is the only defense that seems to have achieved success on `Wa-kNN`, but at a large bandwidth and time overhead cost. Nevertheless, the defense is limited; some other attack could achieve a higher accuracy than `Wa-kNN` on Decoy.

The MAA of all of these defenses are very close to 1 for our data set, since they are limited.

We see that the situation with limited defenses is somewhat dire. While many of those defenses served to defeat previously known attacks (Adaptive, Morphing and HTTPOS demonstrated success against `Li-NBayes`), we see that `Wa-kNN` performs reasonably well against them. We can continue to construct limited defenses that defeat `Wa-kNN` as well: in Section 5.5.6.1 we will see that dummy packets chosen at the correct location can drop `Wa-kNN` accuracy to 6% at only 25% bandwidth overhead. But we cannot recommend such a defense, due to our observation that limited defenses always eventually fail. We can never confidently recommend limited defenses to clients for privacy protection.

5.5.3 Tamaraw Evaluation

In Section 5.2.2.2, we described Tamaraw, a defense that seeks to be effective as a general defense and efficient in terms of bandwidth overhead. Tamaraw builds upon BuFLO to achieve solid privacy guarantees. Tamaraw has five parameters:

1. ℓ_{ppad+}, the size to which all outgoing packets are padded (or split and then padded) for packet padding.

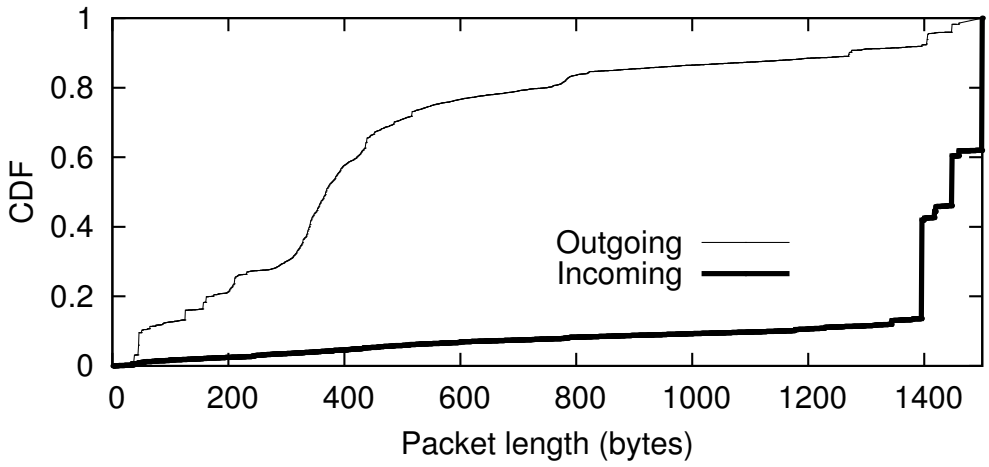

Figure 5.1: CDF of packet lengths observed when loading Alexa's top 100 sites.

2. ℓ_{ppad-}, the above, for incoming packets.
3. ρ_+, the interpacket timing for outgoing packet scheduling.
4. ρ_-, the above, for incoming packets.
5. L_{seg}, the number of cells to pad to for sequence padding.

In this section, we evaluate how the choice of these parameters affect the bandwidth overhead O_D^B, the time overhead O_D^T, and the MAA for Tamaraw. In doing so, we compare with BuFLO to demonstrate that our modifications indeed significantly improve the defense.

5.5.3.1 Packet padding ($\ell_{ppad+}, \ell_{ppad-}$)

When operating on the non-Tor data set, we pad outgoing packets to a length of ℓ_{ppad+} and incoming packets to ℓ_{ppad-} instead of 1500 to reduce the bandwidth overhead. To identify why this is the case, we want to identify the distribution of packet lengths in the non-Tor data set. We present the CDF in Figure 5.1. We see that outgoing packets are concentrated in the 300–600 byte range; this byte range accounted for 46% of our data. We can see that padding all of them to size 1500 will unnecessarily increase bandwidth overhead. Incoming packets are relatively evenly distributed except for several spikes close to 1500 bytes, because the MTU was at 1500 bytes.

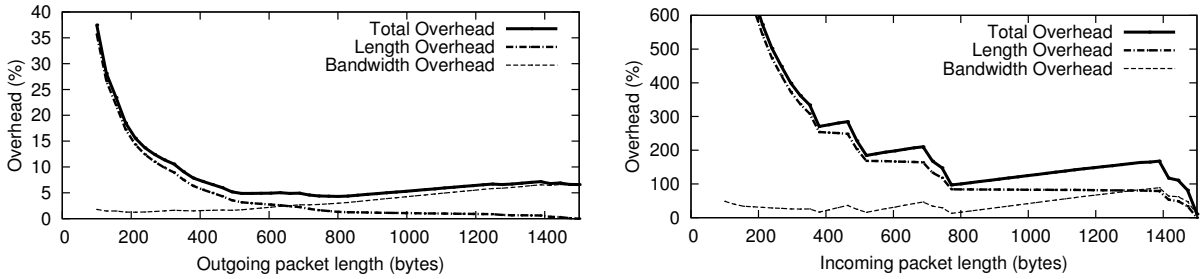

Figure 5.2: Tamaraw: Bandwidth and length overhead while varying ℓ_{ppad+} (left) and ℓ_{ppad-} (right) across Alexa's top 100 sites. The top thick line represents the sum of bandwidth and length overhead. Note that the two graphs have different y-axes.

From the same data set in Figure 5.1, we derive the overhead of choosing ℓ_{ppad+} and ℓ_{ppad-}. We consider two types of overhead:

Bandwidth overhead (O_D^B). The bandwidth overhead includes two components: padded zeroes to pad all packet sizes to ℓ_{ppad+} and ℓ_{ppad-}, and 54 bytes of overhead if there is a packet split (14 bytes of overhead for the Ethernet MAC header, 20 bytes of overhead for the TCP header, and 20 bytes of overhead for the IP header).[12]

Length overhead (O_D^L). The length overhead is the number of packets in the defended sequence divided by the original number of packets. More packets put more weight on routers, such as CPU computation for encryption and decryption. The length overhead is related to the time overhead: when there are more packets, the same packet scheduling parameters will take correspondingly more time to send all packets. Divided packets must necessarily wait for the full duration between time windows, increasing time overhead. However, the reader should note that length overhead is not equivalent to the time overhead. The exact time overhead depends on the packet scheduling parameters, which we will discuss later; a change in the total number of packets (length overhead) might change the optimal parameters for time overhead, such that the length overhead itself is not sufficient to determine the time overhead.

We plot the results for outgoing packets and incoming packets in Figure 5.2. We can see that the situation for outgoing packets and incoming packets are very different; we discuss those two separately as follows.

[12] Note that while this setup is the most common amongst desktop users, it does not necessarily generalize to all users.

Outgoing (ℓ_{ppad+}). For outgoing packets, bandwidth overhead increases and length overhead decreases with increasing ℓ_{ppad+}. Bandwidth overhead increases when a greater ℓ_{ppad+} increases padding within each packet; length overhead decreases as a higher ℓ_{ppad+} splits packets less often. The sum of bandwidth and length overhead decreases and then increases, reaching a minimum at $\ell_{ppad+} = 800$, with $O_D^B = 2.9\%$ and $O_D^L = 1.3\%$. If we only optimize for bandwidth overhead, the minimum is at $\ell_{ppad+} = 210$, with $O_D^B = 1.2\%$ and $O_D^L = 14\%$. This is compared to $O_D^B = 6.5\%$ and $O_D^L = 0\%$ at $\ell_{ppad+} = 1500$.

Incoming (ℓ_{ppad-}). With incoming packets, the lines in the graph are not smooth. Rather, they appear as if they are composed of several distinct lines. The reason for this is as follows: almost all incoming packets have the MTU size 1500, so the optimal points for bandwidth overhead occur when for some integer n, $n \cdot \ell_{ppad-} = 1500$, e.g. at $\ell_{ppad-} = 750$ or $\ell_{ppad-} = 500$. In addition, there is almost no decrease in length overhead when increasing ℓ_{ppad-}, except at those points. Whether we optimize for bandwidth overhead or the sum of bandwidth and length overhead, the optimal is simply padding to $\ell_{ppad-} = 1500$, with $O_D^B = 10\%$.

From these results, we choose the value $\ell_{ppad+} = 800$ and $\ell_{ppad-} = 1500$ for further experiments. Note that from this point onwards in this section, we cease using the non-Tor data set, and perform all experiments on the Tor data set, for which packet padding is not necessary; the results above are only relevant to the non-Tor data set.

5.5.3.2 Packet scheduling (ρ_+, ρ_-)

In BuFLO, ρ_+ and ρ_- were both 0.02 s, but we expect that ρ_+ should be larger than ρ_- because there are fewer outgoing packets in the same amount of time. As whether or not two packet sequences collide is controlled by the padding parameter L_{seg}, our objective in the choice of ρ_- and ρ_+ is to minimize overhead. We test the bandwidth and time overhead on the 100×10 Tor data set. We vary ρ_+ and ρ_- from 0.005 s to 1 s exponentially, taking 30 possible values for ρ_+ and ρ_- each, and thus 900 data points in total.

We present the results in Figure 5.3. Note that this figure only counts the overhead from packet scheduling and it is less than the final overhead, which includes that of sequence padding. We see that in fact there is a significant benefit to choosing the correct ρ_+ and ρ_-. Furthermore, we see that many of the Pareto-optimal points lie on one line: the line represents the data points for which $\rho_+ = 4.0\rho_-$ in our data set. This is expected, due to the fact that there are many more incoming cells than outgoing cells: in fact, there are around 10 times as many. The fact that the multiplier is only 4.0 and not 10 implies that the ratio of fake outgoing cells to real outgoing cells

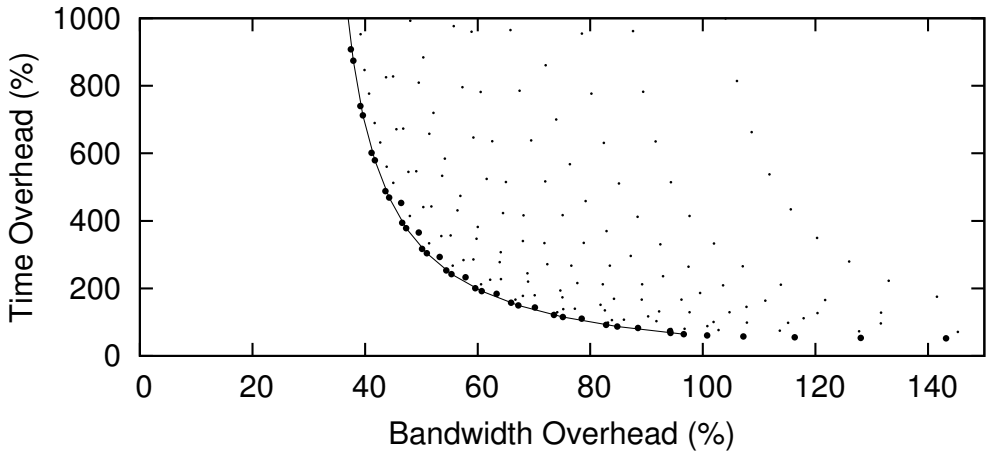

Figure 5.3: Tamaraw: Bandwidth and time overhead while varying ρ_+ and ρ_- from 0.005 s to 1 s across Alexa's top 100 sites. The light dots represent combinations of ρ_+ and ρ_-. The thicker dots represent the Pareto optimal, for which no other point had both a better bandwidth and time overhead. The thick line represents the line for which $\rho_+ = 4.0\rho_-$.

would be much higher than that for incoming cells; we send outgoing cells as soon as possible as an implicitly learned strategy because a single late outgoing cell could delay many incoming cells.

If we were working on a non-Tor data set, Pareto-optimal choices of ρ_+ and ρ_- would be about three times larger because TCP packets are about three times larger than Tor cells.

Any Pareto-optimal point may be the optimal point depending on the client and the network's needs; some clients may tolerate a higher time overhead and some might not. Selecting the minimum of the sum of time and bandwidth overhead, we choose $\rho_+ = 0.020$ s and $\rho_- = 0.005$ s. At this rate the bandwidth overhead is 97% and the time overhead is 64%. We move on to sequence padding with the above choice, keeping in mind that we can trade off an increase in bandwidth overhead for a lesser decrease in time overhead and vice versa.

5.5.3.3 Sequence padding (L_{seg})

Tamaraw significantly improves the sequence padding mechanism of BuFLO. Tamaraw pads both outgoing and incoming sequence length to multiples of L_{seg}, while BuFLO pads to a single constant L_{max} and performs no sequence padding if $|P| > L_{max}$. Here, we investigate the MAA induced by choices of L_{seg}.

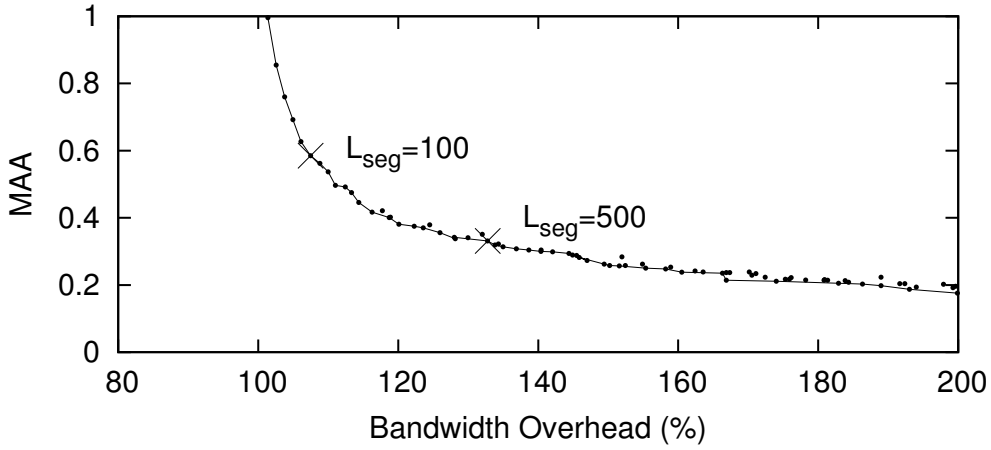

Figure 5.4: Tamaraw: Maximum Attacker Accuracy versus bandwidth overhead while varying L_{seg} from 10 to 1000 in the closed world. We plot a line to show the Pareto-optimal points.

We vary L_{seg} from 1 to 1000, and plot the results in Figure 5.4. We take the 100×10 data set on Tor, thus focusing only on the TPR.[13] The overhead includes that of packet scheduling, which is $O^B = 97\%$ and $O^T = 64\%$ with the parameters $\rho_+ = 0.020$ s and $\rho_- = 0.005$ s, the optimal parameters we found in Section 5.5.3.2.

We observed that the bandwidth and time overhead both increase linearly as L_{seg} increases. For the bandwidth overhead, this mimics the result of Lemma 4. The time overhead also increases linearly because we use fixed-rate packet scheduling, in which case the relationship between the time overhead and the bandwidth overhead is linear. Varying L_{seg} from 1 to 1000, the bandwidth overhead O^B increased from 97% to 191% and the time overhead O^T increased from 64% to 175%. We can see that even a small choice of L_{seg} is sufficient to vastly decrease the MAA: at $L_{seg} = 100$, $a^M = 0.59$ on our data set, with $O^B = 107\%$ and $O^T = 83\%$.

It should be noted that 83% time overhead is very large: the mean time to load a page will become 28 seconds on our Tor data set. Furthermore both BuFLO and Tamaraw have the unfortunate effect of spreading out page loading evenly across time. In normal web browsing, content delivery is uneven: On average, the first quarter of each packet sequence by time contains only 13% of all packets. Tamaraw's constant-rate packet scheduling mechanism cannot adjust for such a phenomenon.

Next, we perform the same experiment in the open world, adding 1000 non-monitored elements to make $100 \times 10 + 1000$, and plot the results in Figure 5.5. We plot both the TPR and

[13] The number of instances does not significantly change experimental results, unlike attacks.

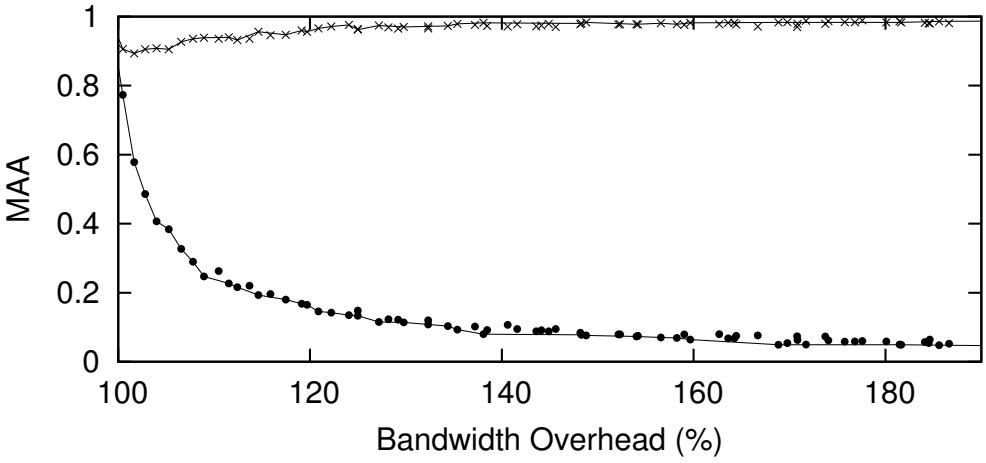

Figure 5.5: Tamaraw: Maximum Attacker Accuracy versus bandwidth overhead while varying L_{seg} from 10 to 1000 in the open world. Crosses represent the TNR and dots represent the TPR. We plot lines to show the Pareto-optimal points.

the TNR. Note that as discussed in Section 5.1.4.1, at every level of L_{seg} it is possible for the perfect attacker to trade a lower TPR for a higher TNR and vice-versa; we only use the non-biased strategy of guessing the most frequently occurring class in each collision set. We can see that the presence of the open-world elements decreases the TPR significantly. At $L_{seg} = 100$, $a^{MTPR} = 0.38$ and $a^{MTNR} = 0.91$. At $L_{seg} = 500$, $a^{MTPR} = 0.12$ and $a^{MTNR} = 0.97$. a^{MTPR} decreases and a^{MTNR} increases with increasing L_{seg}, as the non-monitored class overwhelms every collision set. At higher values of sequence padding bandwidth overhead, the attacker guesses the non-monitored class for almost all collision sets.

We plot the sizes of collision sets in the open-world scenario in Figure 5.6, with $L_{seg} = 100$. We see that the majority of collision sets are very small; 58% of collision sets have size 1 and 94% of them have a size smaller than 10. Note that the CDF in Figure 5.6 considers collision sets, not packet sequences, but the MAA considers packet sequences. While the majority of collision sets have size 1, the majority of packet sequences do not belong to collision sets of size 1. To illustrate this point, we plot Figure 5.7, where the CDF is counted over all packet sequences rather than all collision sets. We see that about 16% of packet sequences are uniquely identifiable within our set of 2000 packet sequences; these packet sequences can be correctly classified by the perfect attacker. While they were uniquely identifiable within our data set, they may still collide with other packet sequences outside of our data set; the perfect attacker must take the possibility of false positives outside of the training set into consideration.

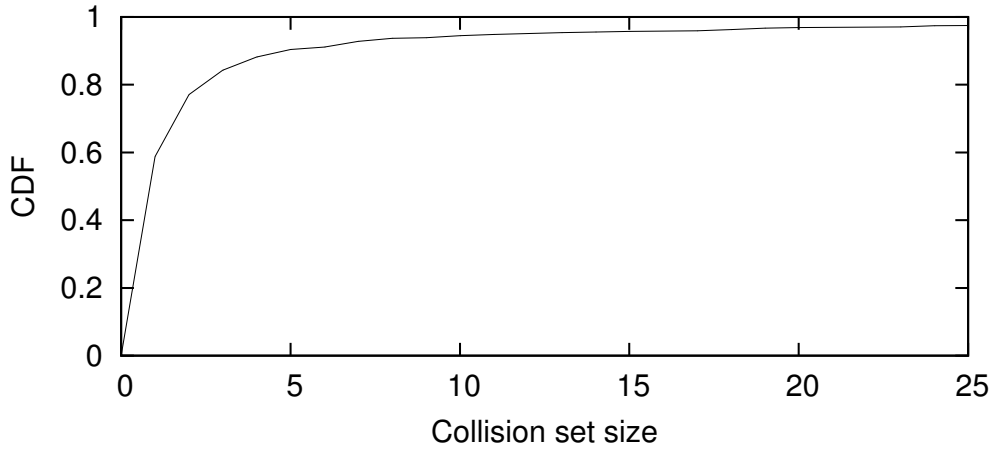

Figure 5.6: Tamaraw: CDF of collision set sizes in the open world with $L_{seg} = 100$, taking the distribution over collision sets.

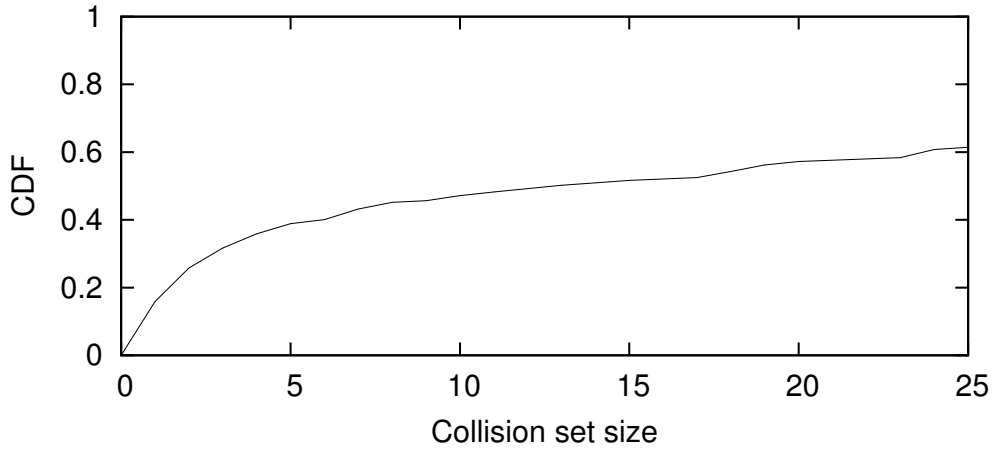

Figure 5.7: Tamaraw: CDF of collision set sizes in the open world with $L_{seg} = 100$, taking the distribution over packet sequences.

5.5.4 Supersequence Evaluation

In this section we present experimental results on Supersequence. Supersequence differs significantly from Tamaraw in its packet scheduling mechanism, as it schedules according to a learned supersequence rather than Tamaraw's constant-rate sequence.

Our results in this section are significantly different from results presented in our previous work [WCN+14]. The difference originates from the fact that we did not consider the time overhead in previous work; rather, we had optimistically assumed that packet scheduling times were always sufficiently large to send packets of the correct direction through. In this work, we recognize that a large time overhead may discourage clients as much as a large bandwidth overhead. To compare Supersequence fairly with other general defenses, we use p_{vote} for packet scheduling to determine packet times, and thus derive its time overhead. We analyze Supersequence in terms of the SCS approximation algorithm, packet scheduling, and sequence padding. Then, we analyze the effect of class-level information on Supersequence.

5.5.4.1 SCS approximation

SCS is an NP-hard problem for which the best known algorithms are exponential, and thus we use an approximation described in Section 5.3.2 to compute the SCS. First, we desire to show that the approximation is fast. We computed the SCS of n randomly chosen input sequences and recorded the time it took to perform this computation, varying n from 20 to 1000.

We plot the results in Figure 5.8 with the standard deviation of 20 trials each as error bars. The graph shows us that the supersequence approximation time is linear in terms of the number of input sequences, increasing up to around half a minute for 1000 input sequences. We derive each element of the supersequence by enumerating through each input packet sequence and tallying their votes, which itself is constant-time, and thus the final time is indeed linear.

Next, we want to know whether or not the SCS approximation is close to the real SCS. We implement the real SCS algorithm on two input sequences, where the computation time is small enough to be feasible. The SCS approximation is necessarily worse than the real SCS. We compare only the number of dummy packets added by either algorithm, setting $p_{vote} = 1$ for our SCS approximation algorithm.

Over the 100×10 Tor data set, we compare these SCS algorithms on random pairs of packet sequences P^1 and P^2. We find that the real SCS algorithm on random pairs of packet sequences adds 2700 ± 3000 dummy packets, whereas our approximation algorithm adds 2900 ± 3100 dummy packets: a small difference across the mean, but a large standard deviation. To arrive at useful conclusions we break this down further. As percentages, the overheads are 53% and

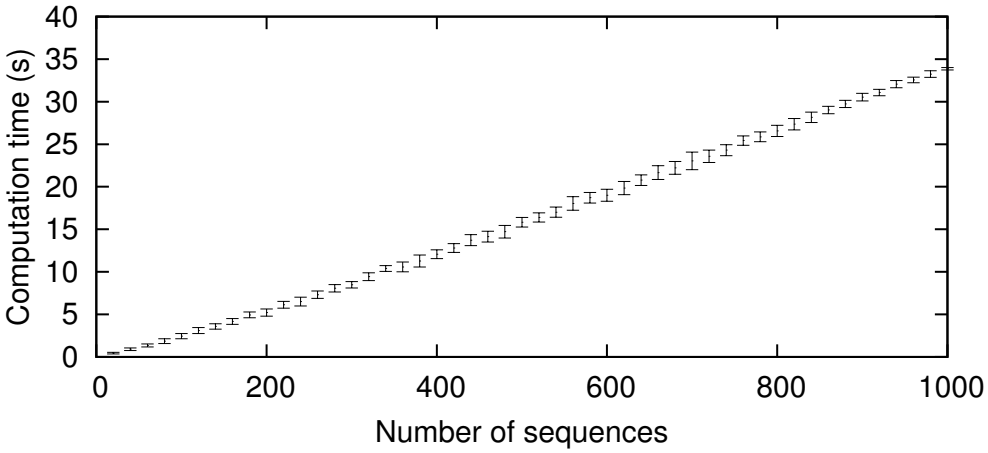

Figure 5.8: Amount of time required to approximate the SCS of a set of input packet sequences, using our simple voting algorithm. We vary the number of inputs from 20 to 1000.

56% respectively. Comparing the SCS of P^1 and P^2 with only the longer (without loss of generality) P^1, we find that the real SCS is only $1.5\% \pm 2.7\%$ longer than $|P^1|$, whereas the SCS approximation is $4.5\% \pm 3.5\%$ longer than $|P^1|$. The difference in overhead between the real and approximate SCS is $3.0\% \pm 1.6\%$. We can compare these values with Decoy, which on average adds 100% overhead to attempt to confuse two random packet sequences with each other.

5.5.4.2 Packet scheduling

As described in Section 5.3, packet scheduling in Supersequence is controlled by a parameter p_{vote}. A larger p_{vote} increases the time gap and thus time overhead between packets, waiting longer to permit more packets to go through, and thus also reducing bandwidth overhead. p_{vote} has a comparable effect to ρ_+ and ρ_- in Tamaraw as plotted in Figure 5.3. On the Tor data set with 100×10, we vary p_{vote} from 0.02 to 1 and plot the bandwidth/time trade off in Figure 5.9, keeping L_{seg} fixed at 1 to disable sequence padding. We obtain 10 values for each setting and plot the standard deviation for both bandwidth and time overhead. For each point of p_{vote} we compute one supersequence on a random size 200 subset of all instances. Packet scheduling cost is more expensive for Supersequence than for Tamaraw. We choose the minimum of the sum of bandwidth and time overhead for Supersequence, which occurs when $p_{vote} = 0.74$, where we have $O^B = 80\% \pm 4\%$ and $O^T = 62\% \pm 10\%$. Note that the standard deviation is very high; we examine this next.

We found that the overhead of Supersequence has a large variance. This is because the su-

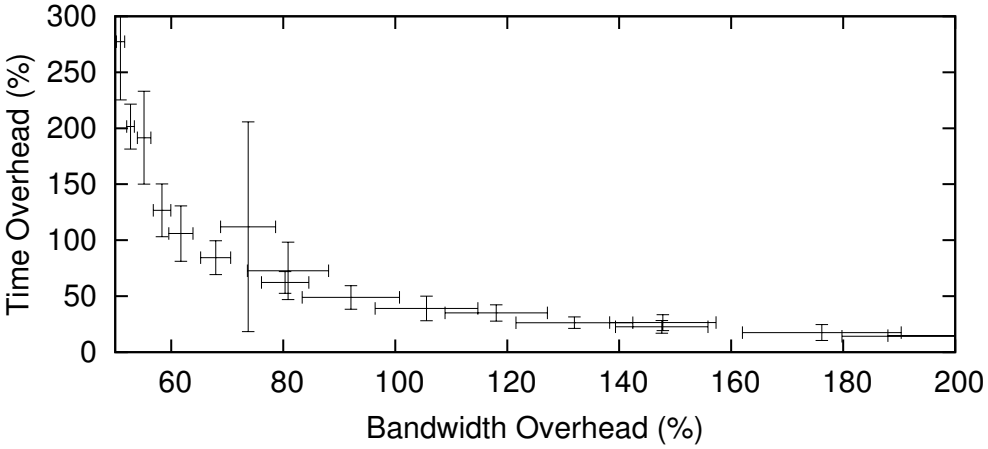

Figure 5.9: Supersequence: Bandwidth and time overhead while varying p_{vote} from 0.02 to 1 across Alexa's top 100 pages. Both x-axis and y-axis error bars are given.

persequence is highly sensitive to the randomly chosen subset of size 200, and poorly chosen supersequence sets can significantly change overhead values. We randomly choose 500 supersequences with $p_{vote} = 0.74$ and plot all observed bandwidth and time overhead values in Figure 5.10. We see that the bandwidth overhead ranges between 70% and 110%, and the time overhead ranges between 40% and 100%. For those 500 supersequences, we have $O^B = 86\% \pm 7\%$ and $O^T = 53\% \pm 14\%$. We do not have a mechanism to identify when the supersequence set is poor and we should choose a new one. Therefore when evaluating Supersequence, we plot graphs with error bars.[14]

5.5.4.3 Sequence padding

Supersequence uses the same sequence padding scheme as Tamaraw, with rounding parameterized by L_{seg}. We vary L_{seg} from 0 to 2000 and plot the results in Figure 5.11, comparable to Figure 5.4. We plot the x-axis error bars (there is very little error in the y-axis) by randomly choosing 10 supersequences between experiments for the data points in Figure 5.11, so that the results also contain the randomness described in Figure 5.10. We see that randomness affects the bandwidth overhead more than L_{seg}, although L_{seg} does steadily decrease a^M (as expected from our theoretical results).

L_{seg} has different effects on Tamaraw and Supersequence. Tamaraw allows the incoming and

[14] We do not plot error bars for Section 5.5.3 because Tamaraw does not train on random packet sequences. Only changes in the data set would affect Tamaraw.

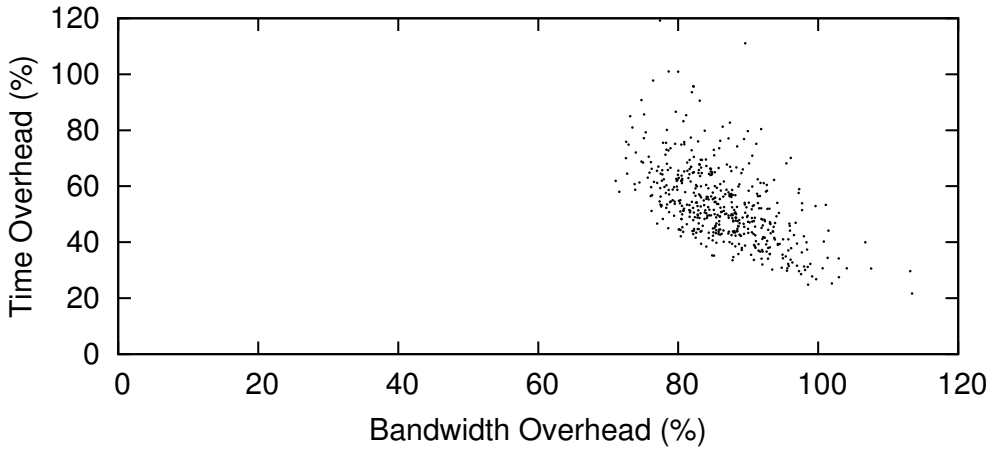

Figure 5.10: Supersequence: Bandwidth and time overhead for 500 supersequences computed from randomly chosen subsets of size 200, with p_{vote} fixed at 0.74 and no sequence padding.

Table 5.5: Several data points of comparison between the effect of L_{seg} on Supersequence and Tamaraw. We denote them as D_S and D_T here. We draw these data points from Figure 5.11 and Figure 5.4, which use the 100×10 Tor data set.

	Tamaraw			Supersequence		
L_{seg}	$O^B_{D_S}$	$O^T_{D_S}$	$a^M_{D_S}$	$O^B_{D_T}$	$O^T_{D_T}$	$a^M_{D_T}$
100	107%	83%	0.59	84%	59%	0.39
500	133%	137%	0.33	93%	60%	0.21
800	150%	174%	0.26	94%	63%	0.17
1500	194%	268%	0.19	98%	67%	0.14

outgoing packets to be rounded to different multiples of L_{seg}, whereas Supersequence does not allow this possibility: the total sequence length is mapped to a multiple of L_{seg}. As such for a given L_{seg} the expected amount of sequence padding is equal to L_{seg} for Tamaraw and $L_{seg}/2$ for Supersequence, and the number of collision sets is also much smaller for Supersequence. We present a table to compare several data points in Table 5.5. We see that Supersequence trumps Tamaraw in every metric, and it scales better with higher L_{seg}, allowing the client to achieve lower a^M with lower overhead.

We will see further methods to reduce Supersequence overhead with higher levels of information next.

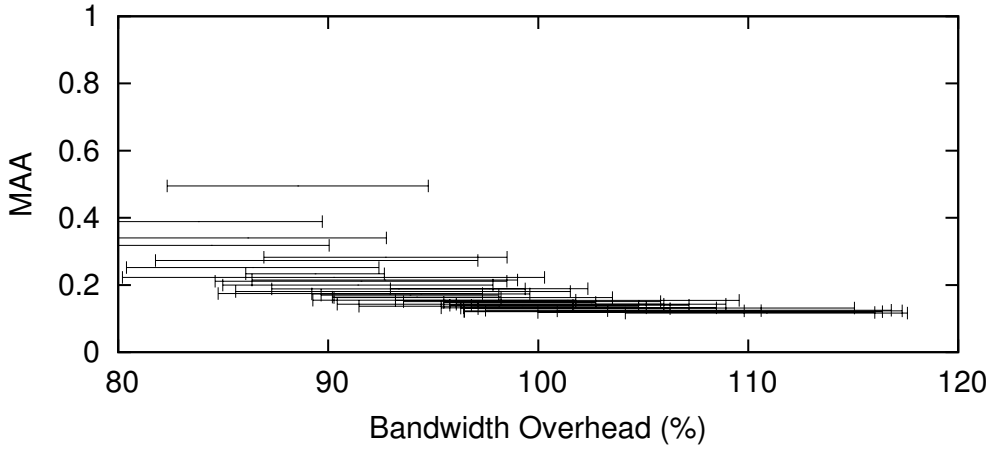

Figure 5.11: Maximum Attacker Accuracy versus bandwidth overhead while varying L_{seg} from 0 to 2000 in the closed world.

5.5.4.4 Class-level information

Under class-level information, different collision sets can have different supersequences. On the other hand, under sequence end information, different collision sets are only different prefixes of the same supersequence. However, increasing the number of supersequences would also increase the number of collision sets, so in the end it may not be worth having several different supersequences at all. We test the clustering scheme described in Section 5.3 to see if it ends up decreasing the overhead and MAA.

For sequence end information, there is one cluster with $p_{vote} = 0.74$ and $L_{seg} = 100$. For class-level information, we vary the number of clusters (equal to the number of supersequences) from 2 to 30. To adjust for the increased number of collision sets, we also multiply L_{seg} by the number of clusters; Lemma 3 shows that this will keep the MAA constant.

We plot the results in Figure 5.12. It appears that clustering gave no significant benefit for the bandwidth and time overhead. In fact, the more clusters there are, the greater the bandwidth and time overhead. In previous work we showed some benefit to clustering with 2 clusters [WCN+14], but that implementation ignored timing and did not take the large standard deviation into consideration. We find no benefit to clustering with class-level information using our clustering algorithm, though it is certainly possible that better clustering algorithms can decrease overhead for class-level information.

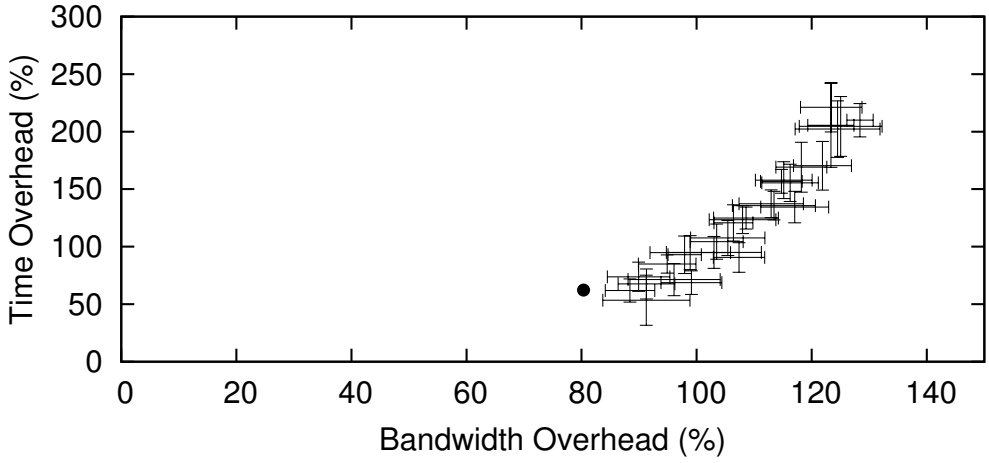

Figure 5.12: Bandwidth and time overhead while varying the number of clusters from 2 to 30 for class-level information. The black dot indicates the bandwidth and time overhead for sequence end information. Increasing the number of clusters increased both the bandwidth and time overhead at the mean. The MAA is approximately the same for all points in this graph.

5.5.5 Walkie-Talkie Evaluation

We implemented half-duplex communication on Tor Browser and obtained a $100 \times 50 + 5000$ data set with it in order to test Walkie-Talkie sequence padding. For all Walkie-Talkie experiments, we use this data set.

In Section 5.4 we described deterministic and random sequence padding in Walkie-Talkie. We test a deterministic padding scheme (`Det`) and three random padding schemes (`Nofake`, `Normal`, and `Uniform`), described below:

Real bursts. `Det` rounds real bursts according to multiples of some L_{seg} much like Tamaraw. For the random scheme, we will pad real bursts by adding a number of packets drawn from some X_{i+} and X_{i-}, which are uniform distributions between 0 and a constant X_{max+} and X_{max-} respectively. Section 5.1.6 suggests that uniform distributions are best.

Fake bursts. `Det` does not add fake bursts. For the random scheme, we will add fake bursts at some probability p_{dummy} after each real burst. For implementation reasons we also limit the maximum number of fake bursts to 9 (otherwise computation time is far too high). The fake bursts themselves are not parameterized, but rather we test three types of fake bursts:

1. `Nofake`: We add no fake bursts.

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.

To find the optimal parameters for the defense, we tested each of these four algorithms in two steps.

1. We randomly generated 10,000 random selections of the parameters above: L_{seg}, X_{max+}, X_{max-}, and p_{dummy} for each of `Det`, `Nofake`, `Normal`, and `Uniform`. These parameters were chosen from uniform distributions across their relevant ranges. Then, we obtained the MAA for 100 random packet sequences and the overhead for all packet sequences. Obtaining the MAA is computationally expensive because the attacker needs to evaluate all possible combinations of real and fake bursts.

2. Out of the 10,000 parameter sets, we chose 100 of the best-performing selections of parameters. Then, we 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 consider an attacker who attempts to minimize his overall MAA, being the proportion of true classifications across both positive (monitored) and negative (non-monitored) instances.

True Positive Rate. First, we focus on the TPR. We obtain the Pareto-optimal results (minimal overhead, minimal TPR) from this list and plot them in Figure 5.13 for each padding algorithm. We see that `Det`, our deterministic algorithm, did not perform well compared to the random algorithms. At a $O^B = 80\%$, a^M was respectively $96.0\% \pm 0.3\%$, $91.0\% \pm 0.8\%$, $92.2\% \pm 0.5\%$ and $80\% \pm 3\%$ for the four algorithms `Det`, `Nofake`, `Normal`, and `Uniform`. Among these algorithms, `Uniform` fake bursts was by far the most efficient, so we will only continue to investigate Walkie-Talkie with `Uniform` fake bursts.

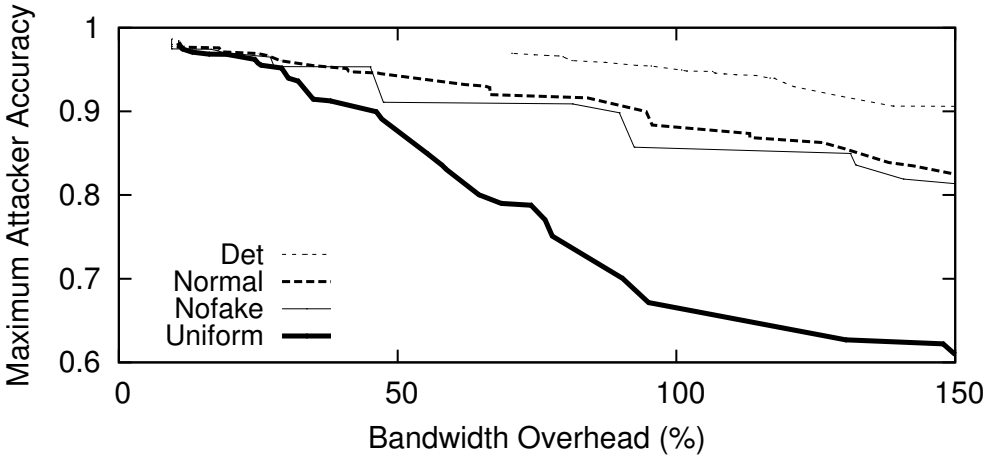

Figure 5.13: MAA (TPR) for the four padding algorithms described in Section 5.5.5. Points are Pareto-optimal in bandwidth overhead and TPR. Note that the y-axis does not start at 0.

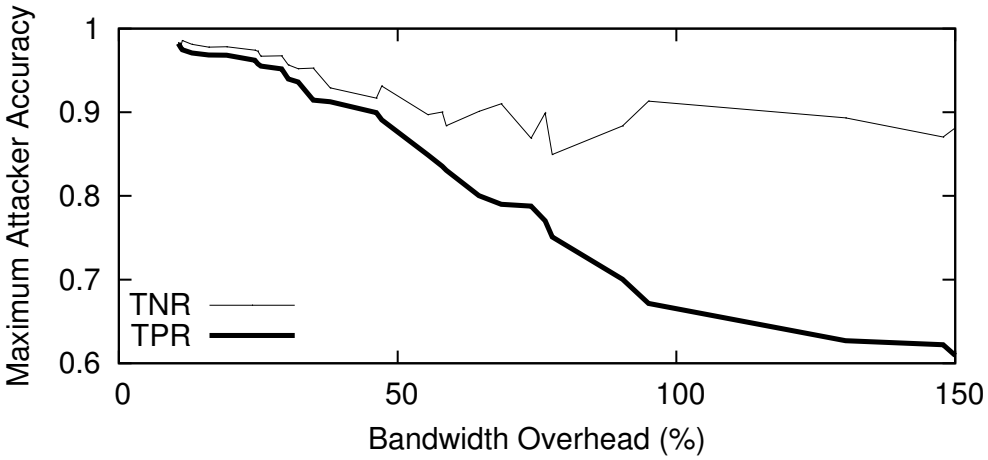

Figure 5.14: MAA (TPR, TNR) with `Uniform` fake bursts. Points are the same as those in Figure 5.13. Note that the y-axis does not start at 0.

True Negative Rate. Now, we examine the TNR induced by Walkie-Talkie with `Uniform` fake bursts. The TNR is significant because the base rate fallacy applies to the perfect attacker as well; a low TNR (high FPR) would significantly hamper the attacker if the base rate is low. We take the same points in Figure 5.13 (minimal overhead and TPR) and plot their TPR and TNR in Figure 5.14. Note that this implies that the TNR is not Pareto-optimal, so that TNR in Figure 5.14 is not monotonic. From Figure 5.14 we see that at an overhead of 35%, FPR exceeds 5%; at an overhead of 55%, FPR exceeds 10%. Walkie-Talkie is able to cause significant confusion to even a perfect attacker with relatively small overhead, especially when a low FPR is sufficient.

5.5.5.1 Performance against attacks

We have shown that Walkie-Talkie performs well against a perfect attacker. It is also interesting for us to measure the effectiveness of Walkie-Talkie against known attacks. We tested the effectiveness of the eight WF attacks against Walkie-Talkie using `Uniform` fake bursts. We do not test all twelve WF attacks in Figure 3.1, but rather only the WF attacks that were once the state of the art. 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, using a 100×40 half-duplex data set. The success rates of the attacks in the open-world scenario is strictly worse than in the closed-world scenario.

We show the results in Table 5.6 under four columns: the original accuracy under the 100×100 Tor data set, the accuracy after packet scheduling (half-duplex communication) but no padding (NoPad), the accuracy with light padding (LightPad) and the accuracy with heavy padding (HeavyPad). LightPad uses `Uniform` fake bursts with $O^B = 23\%$ and $a^M = 0.97$. HeavyPad uses `Uniform` fake bursts with $O^B = 64\%$ and $a^M = 0.84$.

We can see from Table 5.6 that none of the attacks comes close to the perfect attacker's maximum accuracy. `Li-Jac`, `Li-NBayes` and `He-MNBayes` are highly inaccurate against Walkie-Talkie because they rely on unique packet lengths, and there are no unique packet lengths in our scenario. Out of all the attacks, `Pa-FeaturesSVM` appears to suffer least from the padding. Indeed, previous authors [DCRS12, CNW+14] have noted the resilience of this attack against random noise. One possible explanation is that `Pa-FeaturesSVM` is the only attack that uses the well-studied SVM kernel method, which may be able to transform distances between packet sequences in such a way as to identify and ignore dummy packets.

Table 5.6: Accuracy of eight known WF attacks against Walkie-Talkie. The Original accuracy is the accuracy under the closed-world 100×100 scenario we reported in Section 3.4.1. NoPad only applies half-duplex communication and not sequence padding. LightPad uses `Uniform` fake bursts with $O^B = 23\%$ and $a^M = 0.97$. HeavyPad uses `Uniform` fake bursts with $O^B = 64\%$ and $a^M = 0.84$.

Attack	Original	NoPad	LightPad	HeavyPad
Li-Jac [LL06]	0.72	0.01	0.01	0.01
Li-NBayes [LL06]	0.68	0.44	0.28	0.17
He-MNBayes [HWF09]	0.96	0.06	0.06	0.04
Pa-FeaturesSVM [PNZE11]	0.73	0.65	0.51	0.36
Ca-OSAD [CZJJ12]	0.87	0.55	0.27	0.14
Wa-OSAD [WG13b]	0.91	0.68	0.48	0.26
Wa-FLev [WG13b]	0.70	0.51	0.34	0.19
Wa-kNN [WCN+14]	0.91	0.70	0.38	0.11

5.5.6 Alternative Walkie-Talkie Constructions

In Section 5.4.1, 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 MAA. A summary 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 `Wa-kNN`.

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 (much like Adaptive and Morphing). 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. This is equivalent to class-level information under Supersequence.

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

5.5.6.1 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 `Wa-kNN`. We found that `Wa-kNN` was effective against most limited defenses in Section 5.5.2. This approach is similar to how many older defenses proved their effectiveness [SW06, WCM09, LZC+11, PNZE11, DCRS12].

`Wa-kNN` extracts 4226 features from packet sequences (see Table 3.1). Amongst those, we found that features on bursts of packets (contiguous packets in the same direction) are useful for classification. 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 test `Wa-kNN` against the burst-splitting defense, varying the total number of dummy packets, to obtain a tradeoff between bandwidth overhead and accuracy. We re-learned feature weights for WLLCC on the data set of defended packet sequences. We show the results in Figure 5.15. 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, `Wa-kNN` attack accuracy drops to 6% with the same amount of overhead. At around 50% overhead, the accuracy is 1%, the random guessing accuracy. Comparing with the results in Section 5.5.2, we can see that this defense decreases the `Wa-kNN` accuracy with significantly lower overhead.

Although the variant we describe here is highly successful with low overhead, it would perform very poorly against a perfect attacker; it is not a general defense. 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.

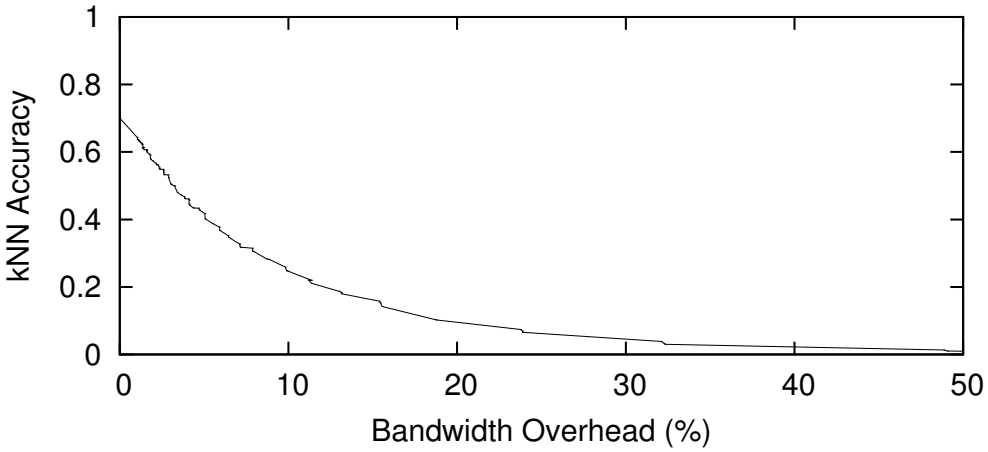

Figure 5.15: `Wa-kNN` accuracy versus overhead with the burst-splitting defense described in Section 5.5.6.1.

5.5.6.2 Without ease of use

We designed Walkie-Talkie 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 as follows.

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 MAA and 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.

5.5.6.3 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; the bandwidth cost of such a query is far smaller than that of web page loading, because the client only needs to load a series of times and directions, rather than packets themselves. Therefore, while expensive to maintain, this variant can significantly save on the client's bandwidth overhead. This setting is similar to the class-level information setting for Supersequence.

Here we use completely random clustering, with no attempt to calculate the distance, on top of half-duplex communication. 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 cause collisions. 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 MAA) in Figure 5.16. With only a 5% overhead, MAA drops below 90%; 25% overhead is enough for MAA to drop below 80%. However, there is a caveat: some pages remain completely

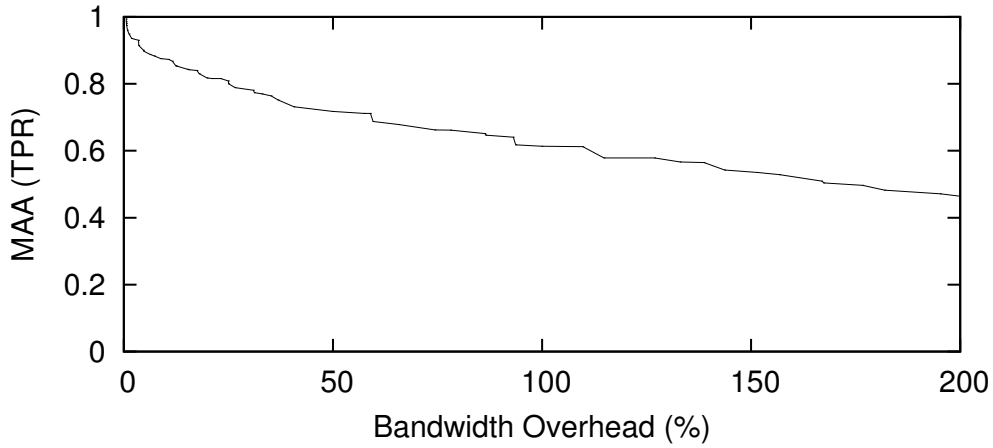

Figure 5.16: Best results for MAA versus overhead when the client queries a central database before packet padding, for Alexa's top 100 sites.

undefended as merging their characteristic sequence with any other would greatly increase the overhead.

5.5.7 General Defenses Comparison

Finally, we take all general defenses: BuFLO, Supersequence, Tamaraw, and Walkie-Talkie, and compare them against each other on the same data set. To compare these defenses fairly, we use the same perfect attacker against all four schemes, and we restrict each defense's capabilities to the following:

1. The defense cannot know the packet sequences it is about to defend beforehand. It may properly parameterize itself by knowing some information about the set of possible packet sequences as a whole, but it has no information that helps it decide which one of the possible packet sequences it will defend next.

2. The defense may use a cooperator who has no information about the packet sequence or any of the packet content, except that the cooperator can identify and drop fake packets.

We present Table 5.7 to show the impact of these restrictions on each defense. Applying these restrictions allows us to compare these defenses fairly. Based on these restrictions, we

Table 5.7: The setting we used for each general defense in order to compare them fairly according to the restrictions set at the start of Section 5.5.7.

Defense	Setting
BuFLO [DCRS12]	The restrictions do not affect BuFLO.
Supersequence [WCN+14]	The restrictions enforce sequence end information on Supersequence.
Tamaraw [CNW+14]	The restrictions do not affect Tamaraw.
Walkie-Talkie [WG15b]	The restrictions enforce the default construction of Walkie-Talkie; we do not use any of the alternative constructions in Section 5.5.6.

give an overall summary of how each defense works in terms of its packet scheduling and sequence padding components in Table 5.8. Table 5.8 specifically chooses the optimal strategies we derived from Sections 5.5.3, 5.5.4 and 5.5.5. We do not need packet padding on Tor, and all experiments in this subsection are performed on Tor.

We proceed to compare each defense by first comparing the packet scheduling component separately, then then adding sequence padding. We do so because the sequence padding of each scheme is quite similar; the core difference is the packet scheduling mechanism. Another advantage is that there are three performance metrics we want to minimize: bandwidth overhead, time overhead, and the MAA. Packet scheduling only affects the bandwidth/time overhead tradeoff, whereas sequence padding only affects the MAA/sequence length tradeoff. Thus, comparing packet scheduling independently from sequence padding allows us to fully investigate the possible tradeoffs in all three parameters.

5.5.7.1 Packet scheduling

We compare the bandwidth and time overhead of each packet scheduling scheme described in Table 5.8, and show the results in Figure 5.17. We describe the results for each defense in detail as follows:

BuFLO. The overhead for BuFLO on our data set was 170% bandwidth overhead and 90% time overhead. This is very different from their results, which was 420% bandwidth overhead and 1.2 s of extra latency on average. The significant difference in these results is due to changes in the data set. Their data set is from Panchenko et al. [PNZE11] which is

Table 5.8: A summary of how each defense applies packet scheduling and sequence padding. More details on each defense are given in Section 5.2.

Defense	Packet scheduling	Sequence padding
BuFLO [DCRS12]	Fixed, constant-rate; ρ, the interpacket time, is $0.0067\ s$ for both incoming and outgoing packets.	$L_{max} = 1500$, so that each packet sequence is padded up to 1500 incoming and outgoing packets.
Tamaraw [CNW+14]	Fixed, constant-rate; we vary ρ_- and ρ_+, the incoming and outgoing interpacket times.	Outgoing sequence length and incoming sequence length are separately padded to multiples of L_{seg}.
Supersequence [WCN+14]	Fixed, variable-rate; the rate is learned from SCS approximation on 200 input packet sequences.	The sequence length is padded to some multiple of L_{seg}.
Walkie-Talkie [WG15b]	No fixed rate; half-duplex communication organizes packet sequences into bursts.	Each burst is separately padded with uniform random padding, with fake bursts added in between.

six years older than ours; web pages and Tor itself have changed significantly since then. For BuFLO to be useful, it must at least have some mechanism to vary and determine the proper interpacket time.

Tamaraw. Tamaraw's bandwidth and time overhead tradeoff is achieved by varying ρ_+ and ρ_-, the outgoing and incoming interpacket times. Since some combinations of ρ_+ and ρ_- are strictly bad (for example, if ρ_- is far higher than ρ_+), we only show the Pareto-optimal results in Figure 5.17. As described in Section 5.5.3, the Pareto-optimal results chiefly consist of points where $\rho_+ = 4.0\rho_-$.

Supersequence. Supersequence's bandwidth and time overhead tradeoff is achieved by varying p_{vote}, a parameter in the SCS approximation voting scheme. Increasing p_{vote} increases the time overhead but decreases the bandwidth overhead. Compared to Tamaraw, Supersequence appears to be generally cheaper in packet scheduling. However, Tamaraw's minimum possible bandwidth overhead is lower than Supersequence: by setting a very high interpacket time, we can reduce the bandwidth overhead of packet scheduling (which is most of the defenses' bandwidth overhead) close to 0 as Tamaraw will no longer need to send any fake packets. But even with $p_{vote} = 1$, Supersequence will still send many fake packets, as the interpacket time it chooses is only the maximum of all observed queue times.

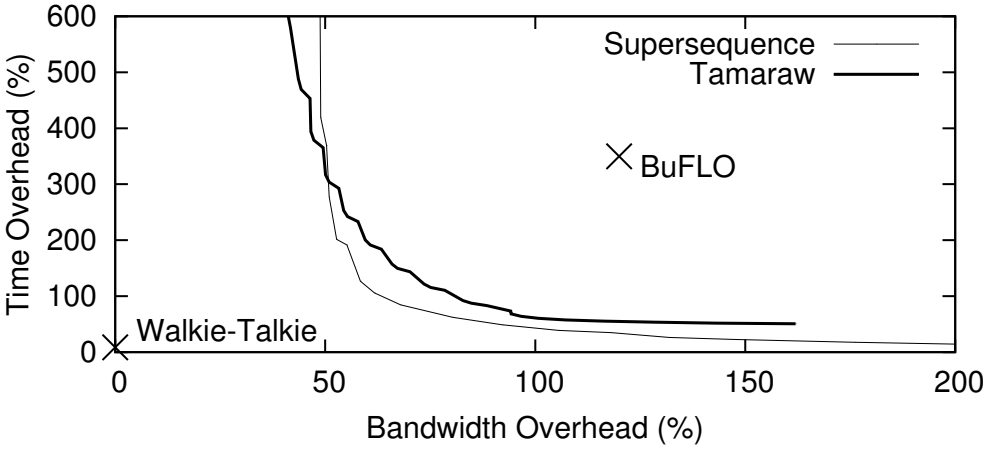

Figure 5.17: Bandwidth and time overhead for the packet scheduling mechanisms of BuFLO, Tamaraw, Supersequence and Walkie-Talkie. For BuFLO and Walkie-Talkie, packet scheduling overhead is not variable.

Walkie-Talkie. Walkie-Talkie has almost no overhead at the packet scheduling stage: there are no added packets and time overhead is only 8.5%. This is the chief advantage of half-duplex communication: by avoiding fixed-rate packet scheduling, it does not need to add or delay any packets during packet scheduling. However, the cost is that the result of packet scheduling leaves Walkie-Talkie with a number of burst lengths, which are harder to pad, whereas the other defenses only leave sequence lengths (BuFLO and Supersequence leave the total sequence length, Tamaraw leave the outgoing and incoming sequence length). BuFLO, Tamaraw and Supersequence will find it much easier to perform sequence padding to cover the sequence lengths, although it will be impossible for them to achieve a lower bandwidth/time overhead than that in Figure 5.17.

5.5.7.2 Sequence padding

To evaluate sequence padding for each defense, we take parameters that minimize the minimum *overhead sum* O^S (sum of bandwidth and time overhead as percentages) for each defense in Figure 5.17. Since sequence padding increases both time and bandwidth overhead evenly for each scheme,[15] we do not need to separate bandwidth and time overhead for evaluating sequence padding. Using the overhead sum allows us to evaluate sequence padding independently of

[15] With the exception that it does not increase the time overhead of Walkie-Talkie.

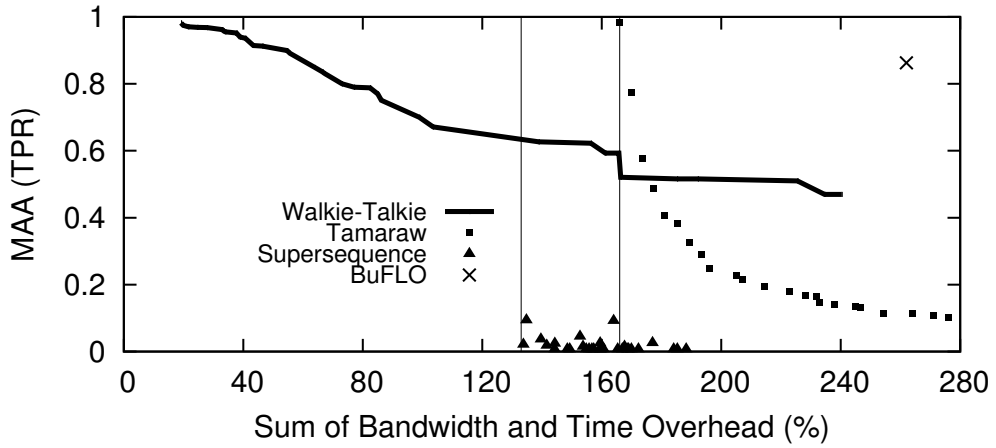

Figure 5.18: Comparison of MAA (TPR) between general defenses versus their sum of bandwidth and time overhead as percentages. The vertical lines represent the minimum overhead sum of Tamaraw and Supersequence, for which we cannot achieve a lower overhead sum on our data set with any parameter selection. Thus the region to the left of the two lines can only be achieved with Walkie-Talkie.

packet scheduling, knowing that we have already minimized the overhead sum. The minimum overhead sum is 156% (99% bandwidth overhead and 67% time overhead) for Tamaraw, and 135% (71% bandwidth overhead and 64% time overhead) for Supersequence. Note that the results in this section represent the *total* overhead of packet scheduling and sequence padding.

We perform experiments on Tor in the open-world $100 \times 10 + 1000$ scenario, and we separate the MAA into TPR and FPR. The perfect attacker attempts to maximize his maximum number of true classifications; since the number of positive and negative elements is the same in our experiments, this is equivalent to maximizing the mean of TPR and TNR.

We plot the results in Figure 5.18 and Figure 5.19 for TPR and FPR respectively. From Figure 5.18, we see that while Tamaraw and Supersequence perform better than Walkie-Talkie in their respective ranges of bandwidth overhead, their minimal overhead is much higher due to packet scheduling. Supersequence, in particular, is effective at high levels of bandwidth overhead. Furthermore, their accuracy is not tunable when their respective packet scheduling parameters are fixed. From Figure 5.19, we see that Walkie-Talkie achieves a good FPR: between overhead sums of 35% to 60%, the FPR increases from 5% to 10%, which is often more than sufficient to overwhelm the base rate of page visits in the sensitive data set. Tamaraw and Supersequence actually have a lower FPR because the TPR is so small that the perfect attacker has resorted to guessing the non-monitored class for almost all collision sets in order to preserve his

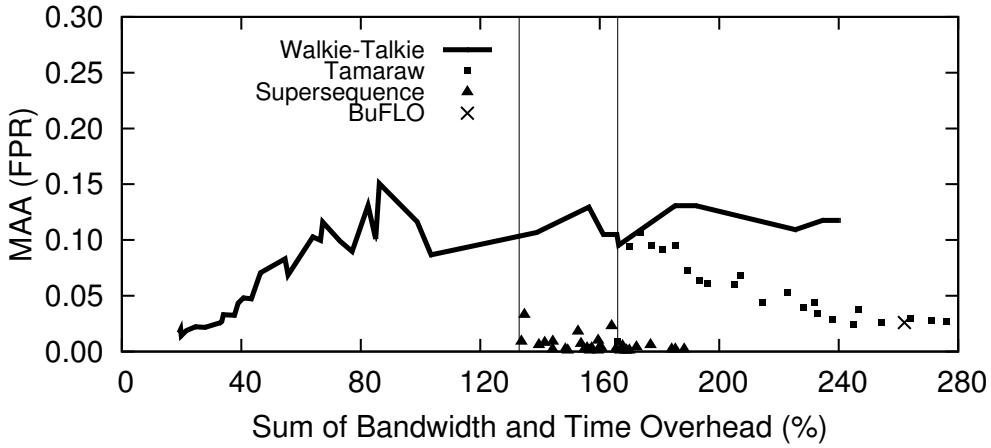

Figure 5.19: Comparison of MAA (FPR) between general defenses versus their sum of bandwidth and time overhead as percentages. The vertical lines represent the minimum overhead sum of Supersequence and Tamaraw, for which we cannot achieve a lower overhead sum on our data set with any parameter selection. Thus the region to the left of the two lines can only be achieved with Walkie-Talkie.

overall accuracy.

5.6 Conclusion

In this chapter, we examined both limited and general defenses. We saw that limited defenses often failed to defeat the latest attacks, specifically `Wa-kNN`. Future website fingerprinting attacks will achieve better results yet on these limited defenses. General defenses protect the client against all possible classification attacks by colliding their packet sequence (deterministically or probabilistically) with that of other web pages.

We described three new general defenses in this chapter: Tamaraw, Supersequence, and Walkie-Talkie.

1. Tamaraw attempts to cut down the overhead of fixed-rate packet scheduling, as used by BuFLO. With several optimizations, we both significantly decrease the overhead and the MAA compared to BuFLO.

2. Supersequence attempts to achieve the optimal deterministic simulatable defense. It selects

packet sequences, possibly with clustering, and uses an approximation algorithm for the shortest common supersequence on this list of packet sequences.

3. Walkie-Talkie significantly reduces the overhead (especially time overhead) of the above schemes by abandoning fixed-rate packet scheduling, instead re-ordering packet sequences into bursts to allow sequence padding.

As general defenses, they defeat all possible attacks and we can evaluate the maximum attacker accuracy of any attack against them.

We uncovered certain principles regarding the relationship between deterministic and random defenses, as well as how MAA and overhead change depending on collision set selection. They served to guide the construction of these defenses. To analyze general defenses effectively, we compartmentalized each general defense into packet padding, packet scheduling, and sequence padding mechanisms. Supersequence and Tamaraw use fixed-rate packet scheduling, which is very expensive in bandwidth and time overhead. Walkie-Talkie uses half-duplex communication for packet scheduling, which is weaker but nearly free in overhead.

Supersequence and Tamaraw, while constructed from different principles, have a similar overhead: the client can expect both her bandwidth and time overhead to increase significantly. The client can then be assured of a large collision set (low MAA). On the other hand, if the client is comfortable with a MAA greater than 0.5, Walkie-Talkie is suitable. Considering that a large MAA could still fall to the base rate fallacy, because an FPR of a few percentage points is enough to conceal most web pages, we argue that Walkie-Talkie is much more practical given that the bandwidth and time overhead represent a degrade in performance to the proxies and the client respectively.

There is certainly significant work to be done in this area, as Walkie-Talkie is not yet ideal. In detail, we point out the following future work:

1. The inefficiency of Walkie-Talkie stems from the large number of bursts required to load each page. In truth, since loading a web page by itself features no interaction from the user, it is entirely possible to load most pages in two bursts (one for the main page HTML, one for the resources required by the page), in which case Walkie-Talkie would be very efficient. It may be possible to change web page loading in some fundamental way to reduce the number of bursts. The only change we made was to increase the number of possible simultaneous connections, which slightly reduces the number of bursts.

2. The perfect attacker model, while theoretically grounded, may be too strong. For example, pages may load random content with random network conditions, but the perfect attacker is

able to identify all possible combinations of randomness and train for them. The maximum attacker accuracy loses its namesake in the open-world scenario, where the perfect attacker may still trade off TPR for TNR and vice-versa, so that there is no definitive maximum attacker accuracy. On the other hand, we have shown evidence that only being able to defend against the state-of-the-art attack is not sufficient. There may be some middle ground between the two extremes that allows for a more useful analysis. For example, we may consider a perfect attacker who knows all the resources on the web page, their lengths, and how they are related (for each resource, which resources it requests). However, he cannot predict network conditions or the random choices within the above; for example, he cannot know exactly which advertisement will be loaded (whereas the perfect attacker we used in our work does). Interestingly, this attacker would not be far from the attack discussed at the end of Section 3.5, where we hypothesized that such an attacker could classify at a higher accuracy.

Chapter 6

Conclusion

In this work, we examined attacks and defenses for website fingerprinting, an application of machine learning to the emergent field of privacy-enhancing technologies. We saw that website fingerprinting attacks threaten the privacy of web-browsing clients: we are able to identify the client's destination server with only passive eavesdropping using machine learning techniques, even if the client is using proxies with encryption (for example, on Tor). We designed several powerful website fingerprinting attacks that achieve a high accuracy in a realistic, open-world scenario. Our latest attack, `Wa-kNN`, is both fast and effective.

To defend against such attacks, the client must apply website fingerprinting defenses. Limited defenses often fail against more powerful website fingerprinting attacks. Instead, we want general defenses that cause collisions between packet sequences to provably defend the client against any possible website fingerprinting attack. We designed several general defenses. On the one hand, if the client desires a general website fingerprinting defense with a very low MAA, she may choose Supersequence or Tamaraw. On the other hand, if the client is comfortable with causing a large false positive rate for the attacker, she may use the low-overhead Walkie-Talkie defense.

References

[ABG12] Mashael AlSabah, Kevin Bauer, and Ian Goldberg. Enhancing Tor's performance using real-time traffic classification. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pages 73–84, 2012.

[AJN+13] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 20th ACM Conference on Computer and Communications Security*, pages 1129–1140, 2013.

[Ale15] Alexa. Alexa — The Web Information Company. www.alexa.com, 2015. Accessed Sep. 2015.

[BLJL06] George Bissias, Marc Liberatore, David Jensen, and Brian Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Privacy Enhancing Technologies*, pages 1–11, 2006.

[BM00] Eric Brill and Robert Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293, 2000.

[CA98] Heyning Cheng and Ron Avnur. Traffic Analysis of SSL-Encrypted Web Browsing. http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps, 1998.

[CB97] Mark Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.

[CC01] Emiliano Casalicchio and Michele Colajanni. A client-aware dispatching algorithm for web clusters providing multiple services. In *Proceedings of the 10th International Conference on World Wide Web*, pages 535–544, 2001.

[CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[CM04] Vladimir Cherkassky and Yunqian Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113–126, 2004.

[CNJ14] Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *Proceedings of the 13th ACM Workshop on Privacy in the Electronic Society*, 2014.

[CNW+14] Xiang Cai, Rishab Nithyanand, Tao Wang, Ian Goldberg, and Rob Johnson. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, 2014.

[COR15] Nate Cardozo, Kurt Opsahl, and Rainey Reitman. Who Has Your Back? Protecting Your Data From Government Requests, 2015. A publication of the Electronic Frontier Foundation.

[CZJJ12] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob 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.

[Dam64] Fred Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3), 1964.

[DCRS12] Kevin Dyer, Scott Coull, Thomas Ristenpart, and Thomas 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.

[Din12] Roger Dingledine. Trip report, ACM CCS/WPES. https://blog.torproject.org/blog/trip-report-acm-ccswpes, December 2012. Accessed Aug. 2015.

[Din14] Roger Dingledine. A call to arms: Helping Internet services accept anonymous users. https://blog.torproject.org/blog/call-arms-helping-internet-services-accept-anonymous-users, August 2014. Accessed Feb. 2015.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[EZ05] Benjamin Edelman and Jonathan Zittrain. Empirical Analysis of Internet Filtering in China. *Berkman Center for Internet & Society, Harvard Law School, http://cyber.law.harvard.edu/filtering/china*, 2005.

[FR14] R. Fielding and J. Reschke. Hypertext transfer protocol (http/1.1): Message syntax and routing. RFC 7230, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7230.txt.

[GH12] Yossi Gilad and Amir Herzberg. Spying in the Dark: TCP and Tor Traffic Analysis. In *Privacy Enhancing Technologies*, pages 100–119, 2012.

[GJH13] John Geddes, Rob Jansen, and Nicholas Hopper. How low can you go: Balancing performance with anonymity in Tor. In *Privacy Enhancing Technologies*, pages 164–184, 2013.

[GKB10] Xun Gong, Negar Kiyavash, and Nikita Borisov. Fingerprinting Websites using Remote Traffic Analysis. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 684–686, 2010.

[Gol03] Ian Goldberg. Privacy-Enhancing Technologies for the Internet, II: Five years later. In *Privacy Enhancing Technologies*, pages 1–12, 2003.

[Gol07] Ian Goldberg. Privacy-Enhancing Technologies for the Internet III: Ten Years Later. In *Digital Privacy: Theory, Technologies, and Practices*. CRC Press, 2007.

[Goo] Google. HTTP Pipelining. https://www.chromium.org/developers/design-documents/network-stack/http-pipelining. Accessed Jul. 2015.

[GR05] Amir Globerson and Sam Roweis. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems*, pages 451–458, 2005.

[GWB97] Ian Goldberg, David Wagner, and Eric Brewer. Privacy-Enhancing Technologies for the Internet. Technical report, DTIC Document, 1997.

[Hin03] Andrew Hintz. Fingerprinting Websites Using Traffic Analysis. In *Privacy Enhancing Technologies*, pages 171–178, 2003.

[HL02] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[HWF09] Dominik Herrmann, Rolf Wendolsky, and Hannes 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.

[JAA+14] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, 2014.

[JL95] Tao Jiang and Ming Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal on Computing*, 24(5):1122–1139, 1995.

[JS12] Suman Jana and Vitaly Shmatikov. Memento: Learning secrets from process footprints. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pages 143–157, 2012.

[JWJ+13] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proceedings of the 20th ACM Conference on Computer and Communications Security*, pages 337–348. ACM, 2013.

[Lab07] The Citizen Lab. Everyone's Guide to By-Passing Internet Censorship. http://www.nartv.org/mirror/circ_guide.pdf, 2007. Accessed Sep. 2015.

[LCC10] Liming Lu, Ee-Chien Chang, and Mun Choon Chan. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *Computer Security–ESORICS 2010*, pages 199–214. Springer, 2010.

[Lev66] Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Physics*, 10(8):707–710, 1966.

[LL06] Marc Liberatore and Brian Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13 ACM Conference on Computer and Communications Security*, pages 255–263, 2006.

[LRWW04] Brian Levine, Michael Reiter, Chenxi Wang, and Matthew Wright. Timing attacks in low-latency mix systems. In *Financial Cryptography*, pages 251–265. Springer, 2004.

[LWD10] Chao Liu, Ryen White, and Susan Dumais. Understanding web browsing behaviors through Weibull analysis of dwell time. In *Proceedings of the 33rd International ACM SIGIR Conference*, pages 379–386, 2010.

[LZC+11] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, and Roberto Perdisci. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *Proceedings of the 18th Network and Distributed Security Symposium*, 2011.

[MCF00] Maurizio Molina, Paolo Castelli, and Gianluca Foddis. Web traffic modeling exploiting TCP connections' temporal clustering through HTML-REDUCE. *IEEE Network*, 14(3):46–55, 2000.

[Moo14] Glyn Moody. NSA's XKeyscore Source Code Leaked! Shows Tor Users Classified As 'Extremists'. https://www.techdirt.com/articles/20140703/02494927769/nsas-xkeyscore-source-code-leaked-shows-tor-users-classified-as-extremists.shtml, July 2014. Accessed Aug. 2015.

[Moz14] Mozilla Developer Network. HTTP Pipelining FAQ. https://developer.mozilla.org/en-US/docs/Web/HTTP/Pipelining_FAQ, 2014. Accessed Jul. 2015.

[OTK76] Teruo Okuda, Eiichi Tanaka, and Tamotsu Kasai. A method for the correction of garbled words based on the Levenshtein metric. *IEEE Transactions on Computers*, 100(2):172–178, 1976.

[Per11a] Mike Perry. Experimental Defense for Website Traffic Fingerprinting. https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting, September 2011. Accessed Aug. 2015.

[Per11b] Mike Perry. TBB's Firefox should use optimistic data socks handshake variant. https://trac.torproject.org/projects/tor/ticket/3875, August 2011. Accessed Aug. 2015.

[Per12] Mike Perry. Implement Adaptive Padding or some variant and measure overhead vs accuracy. https://trac.torproject.org/projects/tor/ticket/7028, Oct 2012. Accessed Aug. 2015.

[Per13] Mike Perry. A Critique of Website Traffic Fingerprinting Attacks. https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks, November 2013. Accessed Aug. 2015.

[Pie03] Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003.

[PNZE11] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas 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.

[Psi15] Psiphon Inc. Psiphon. https://psiphon.ca/, 2015. Accessed Sep. 2015.

[Sco15] David Scott. *Multivariate density estimation: Theory, Practice, and Visualization*. John Wiley & Sons, 2015.

[SSW+02] Qixiang Sun, Daniel R Simon, Yi-Min Wang, Wilf Russell, Venkata N Padmanabhan, and Lili Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 19–30, 2002.

[SW06] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *Computer Security–ESORICS 2006*, pages 18–33. Springer, 2006.

[Tor15] Tor. Tor Metrics Portal. https://metrics.torproject.org/, 2015. Accessed Aug. 2015.

[Tur89] Jonathan Turner. Approximation algorithms for the shortest common superstring problem. *Information and computation*, 83(1):1–20, 1989.

[Uni13] United States Congress. Cyber Intelligence Sharing and Protection Act. https://www.congress.gov/bill/113th-congress/house-bill/624/text, 2013. H. R. 624.

[VC74] Vladimir Vapnik and Alexey Chervonenkis. *Theory of Pattern Recognition*. Nauka, 1974.

[WB90] Samuel Warren and Louis Brandeis. The right to privacy. *Harvard Law Review*, pages 193–220, 1890.

[WBS05] Kilian Weinberger, John Blitzer, and Lawrence Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems*, pages 1473–1480, 2005.

[WCM09] Charles Wright, Scott Coull, and Fabian Monrose. Traffic Morphing: An Efficient Defense against Statistical Traffic Analysis. In *Proceedings of the 16th Network and Distributed Security Symposium*, pages 237–250, 2009.

[WCN+14] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[WG13a] Tao Wang and Ian Goldberg. Comparing website fingerprinting attacks and defenses. Technical Report 2013-30, CACR, 2013. http://cacr.uwaterloo.ca/techreports/2013/cacr2013-30.pdf.

[WG13b] Tao Wang and Ian Goldberg. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society*, pages 201–212, 2013.

[WG15a] Tao Wang and Ian Goldberg. On Realistically Attacking Tor with Website Fingerprinting. Technical Report 2015-08, CACR, 2015. http://cacr.uwaterloo.ca/techreports/2015/cacr2015-08.pdf.

[WG15b] Tao Wang and Ian Goldberg. Walkie-Talkie: An Effective and Efficient Defense against Website Fingerprinting. Technical Report 2015-09, CACR, 2015. http://cacr.uwaterloo.ca/techreports/2015/cacr2015-09.pdf.

[WMMY12] Ronald Walpole, Raymond Myers, Sharon Myers, and Keying Ye. *Probability and statistics for engineers and scientists*. Prentice Hall, 9th edition, 2012.

[XJRN02] Eric Xing, Michael Jordan, Stuart Russell, and Andrew Ng. Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems*, pages 505–512, 2002.

Appendix A

Reproducibility

To ensure reproducibility of our results, we publish the following:[1]

1. Our implementations for all twelve website fingerprinting attacks and eight website fingerprinting defenses. All of them operate on the same data format and can be run side-by-side for comparison. We note that our implementation of other authors' attacks and defenses may be different from their implementations, with an exception for `Ca-OSAD` since we have their original code.

2. All data sets used in this work. This includes the Tor and non-Tor data sets, the splitting data set, and the Walkie-Talkie half-duplex data set.

3. Code for modifying the browser. This includes code to enable half-duplex communication and code to log Tor cells for ground truth in splitting.

We include documentation for all code.

[1] The code is available at https://crysp.uwaterloo.ca/software/webfingerprint/