Zero-delay Lightweight Defenses against Website Fingerprinting

Jiajun Gong and Tao Wang, *Hong Kong University of Science and Technology*

https://www.usenix.org/conference/usenixsecurity20/presentation/gong

This paper is included in the Proceedings of the
29th USENIX Security Symposium.
August 12–14, 2020
978-1-939133-17-5

Open access to the Proceedings of the
29th USENIX Security Symposium
is sponsored by USENIX.

Zero-delay Lightweight Defenses against Website Fingerprinting

Jiajun Gong, Tao Wang
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
{jgongac, taow}@cse.ust.hk

Abstract

Website Fingerprinting (WF) attacks threaten user privacy on anonymity networks because they can be used by network surveillants to identify the webpage being visited by extracting features from network traffic. A number of defenses have been put forward to mitigate the threat of WF, but they are flawed: some have been defeated by stronger WF attacks, some are too expensive in overhead, while others are impractical to deploy.

In this work, we propose two novel zero-delay lightweight defenses, FRONT and GLUE. We find that WF attacks rely on the feature-rich trace front, so FRONT focuses on obfuscating the trace front with dummy packets. It also randomizes the number and distribution of dummy packets for trace-to-trace randomness to impede the attacker's learning process. GLUE adds dummy packets between separate traces so that they appear to the attacker as a long consecutive trace, rendering the attacker unable to find their start or end points, let alone classify them. Our experiments show that with 33% data overhead, FRONT outperforms the best known lightweight defense, WTF-PAD, which has a similar data overhead. With around 22%–44% data overhead, GLUE can lower the accuracy and precision of the best WF attacks to a degree comparable with the best heavyweight defenses. Both defenses have no latency overhead.

1 Introduction

As people increasingly use the Internet for work and entertainment, network surveillance has correspondingly grown to become a pervasive threat against people's privacy. Tor, an anonymity network based on onion routing [21], has become one of the most popular privacy enhancing technologies by defending web-browsing users from network eavesdroppers. To do so, it forwards user packets across multiple volunteer proxies, so that network surveillants cannot see both the true source and destination of the packets.

In the last decade, multiple studies [1, 7, 16, 17, 18, 20, 24, 25, 26, 29, 30] have shown that Tor is vulnerable to *Website Fingerprinting* (WF), a kind of traffic analysis attack where a local attacker passively eavesdrops on network traffic to find out which webpage a client is visiting. WF attackers succeed by observing packet patterns such as the number of outgoing and incoming packets, packet rates, packet timing, and the ordering of packets. (WF attacks do not need to break encryption.) What makes WF attacks especially threatening is that the local passive eavesdropper (which could be the client's ISP) is virtually impossible to detect.

To counter WF attacks, a number of defenses [2, 6, 11, 17, 19, 24, 27, 28] have been proposed over the years, but none have been adopted by Tor or any other privacy enhancing technology. This is because their data overhead may be too high; they may delay packets too much, hurting user experience; they may be too hard to implement realistically, relying on extra infrastructure that cannot be provided; or they may simply be ineffective against the best attacks. A defense against the WF problem grows increasingly urgent as more powerful attacks are found.

Our work makes the following contributions:

1. Emphasizing costlessness, practicality and usability, we design two new defenses that can defeat the best WF attacks: FRONT and GLUE. We call them zero-delay lightweight defenses, meaning they do not delay the client's packets and they only add a small number of dummy packets to real traffic.

    - FRONT obfuscates the feature-rich front portion of traces, which is crucial to the attacker's success. It does so using randomized amounts of dummy packets, disrupting the attacker's training process.
    - GLUE adds dummy packets between traces to make it seem as if the client is visiting pages consecutively without pause. This forces the attacker to solve difficult splitting problems, which previous work finds that even the best attacks fail to do [10].

2. We conduct extensive experiments to show the effectiveness of our defenses. We show that FRONT is able to

outperform WTF-PAD (the previous best zero-delay defense) with the same data overhead (33%) in terms of attackers' performance as well as information leakage analysis, while GLUE can reduce the TPR and precision of the best WF attacks down to single digits with 22%–44% data overhead (overhead depending on user behavior).

3. As GLUE relies on the difficulty of the splitting problem, we improve known solutions to splitting with a new framework, CDSB, to evaluate GLUE fairly. To the best of our knowledge, this is the first work that presents the performance of WF attacks when more than two webpages are visited consecutively.

We organize the rest of the paper as follows. We first discuss the related work in Section 2, and then we give some preliminaries in Section 3. We present FRONT and its evaluation in Sections 4 and 5 respectively, and we present GLUE and its evaluation in Sections 6 and 7 respectively. Finally we summarize our work in Section 8.

2 Related Work

Website Fingerprinting Attacks. WF attacks date back to 2002, when Hintz showed preliminary success in fingerprinting webpages by the number of bytes received in each connection [9]. Later, more studies successfully applied attacks against single-hop systems (Stunnel, OpenSSH, CiscoVPN and OpenVPN) in the closed-world scenario [8, 13]. (We will define the closed-world scenario and the more realistic open-world scenario in Section 3.) These attacks failed to defeat Tor because of Tor's cell-level padding [8]. In 2011, Panchenko et al. [17] showed success against Tor (73% accuracy) with the use of a support vector machine (SVM) using expert features; it was effective in a preliminary open-world scenario as well. Further works [1, 4, 7, 16, 18, 20, 24, 25] have been proposed since then that pushed accuracy higher and false positive rate lower.

We pick four of the best, most recent attacks to evaluate, all of which are highly effective in the open-world scenario:

- **kNN** [24]: Proposed by Wang et al. in 2014, this attack uses a k-nearest neighbors classifier based on automatically learning weights of different features. It is designed to break WF defenses, as it adjusts to defensive feature scrambling by lowering the weights of bad features.

- **CUMUL** [16]: Panchenko et al. proposed this SVM classifier that exploits the "cumulative representation" of a trace in 2016. It is more accurate than kNN, and has an excellent computation time.

- **kFP** [7]: In 2016, Hayes and Danezis proposed this attack that jointly uses random forests and k-nearest neighbors. It has high precision in the open-world scenario.

- **DF** [20]: DF is a recent attack using a deep Convolutional Neural Network. It outperforms other deep learning attacks [1, 18], achieving high precision and recall. It is the first attack shown to be effective against WTF-PAD, a lightweight WF defense [11].

Website Fingerprinting Defenses. To defend against local, passive WF attackers, WF defenses can be deployed on an anonymity network to modify how the client talks to the network's proxies. This is generally done by adding dummy packets or delaying real packets according to some strategy; the attacker cannot distinguish between dummy packets and real packets. No modification to the web server is required. Over the years, researchers have put forward a number of defenses to protect privacy-sensitive clients against WF attacks. We classify the strategies they use to defeat WF attacks into three categories, roughly in order of overhead: obfuscation, confusion, and regularization.

Obfuscation defenses seek to obfuscate specific features WF attacks rely on. A number of early defenses obfuscate packet lengths to defeat older WF attacks. These include *Traffic Morphing* by Wright et al. [28], which pads and splits packets, and *HTTPOS* [14], which does the same on specific HTTP requests and responses. These two defenses are ineffective on Tor, where packet lengths already leak no information because of constant-size cell-level padding. In 2016, Juarez et al. [11] introduced WTF-PAD, which uses a sophisticated token system to generate dummy packets and fill up abnormal trace gaps.

Some defenses aim to achieve *confusion*: they make it difficult for an attacker to determine which of a certain set of given traces is loaded. Panchenko et al. suggested simply loading a *Decoy* page for every true page load [17], so the attacker does not know which is the real page. Wang et al. proposed confusing the attacker by sending two or more traces under a *Supersequence* [24] that is created by adding dummy packets at the right places and delaying user packets.

Much work has been done on *regularization* defenses recently, which restrict how clients can send and receive packets in order to strictly limit the feature space available to the attacker. Some of these defenses enforce a fixed packet rate, with regular sequence end times, on the client: these include BuFLO (Buffered Fixed-Length Obfuscation) by Dyer et al. [6], CS-BuFLO (Congestion-Sensitive BuFLO) by Cai et al. [2], and the overhead-optimized Tamaraw by Cai et al. [3]. Fixing the packet rate delays user traffic significantly. In 2017, Wang and Goldberg [27] introduced Walkie-Talkie, which forces the browser to communicate in half-duplex mode to limit features. It achieves regularization at a lower overhead if we can assume that the client has some knowledge of webpage sizes.

Surveying the extensive work done on confusion and regularization defenses, we find that almost all of them have either a high data overhead (requiring many dummy pack-

Table 1: Comparison of known WF defenses. For overhead, Low is a non-zero overhead up to 35%, Medium is roughly 35–70%, High is roughly 70-100%, and Very High is above 100%.

Category	Defense	Latency overhead	Data overhead	Requires additional infrastructure	Defeated by known attacks
Obfuscation	Traffic morphing [28]	None	Low	None	Yes
	HTTPOS [14]	None	Low	None	Yes
	WTF-PAD [11]	None	Low	None	Yes
	FRONT (this work)	**None**	**Low**	**None**	**No**
Confusion	Decoy [17]	None	High	None	No
	Walkie-talkie [27]	Medium	Low	Knowledge of pages, half-duplex	No
	Supersequence [24]	High	Very High	Knowledge of pages	No
Regularization	BuFLO [6]	Very High	Very High	Fixed-rate network transfer	No
	CS-BuFLO [2]	Very High	Very High	Fixed-rate network transfer	No
	Tamaraw [3]	High	High	Fixed-rate network transfer	No
	GLUE (this work)	**None**	**Low**	**None**	**No**

ets) or cause significant delays to user traffic; sometimes both. These factors have stymied the adoption of all of these defenses; Tor developers would not want to harm user experience of their anonymity network. Therefore, to create zero-delay lightweight defenses, we decided to avoid confusion and regularization defenses. Among our new defenses, FRONT is an obfuscation defense, while GLUE is in its own category as it forces the WF attacker to solve a different, much more difficult problem.

Some other defenses also require extra infrastructure to support, which is detrimental to their deployability. Supersequence and Walkie-Talkie both assume that the client knows some information about the webpage they are about to visit. This is generally impractical. The BuFLO-series of defenses mandate fixed packet rates, which may require some modification to the network stack because otherwise network delays could still reveal information. Walkie-Talkie requires modification to how the browser loads webpages. Our objective is to create defenses that can be deployed as painlessly and quickly as possible against the present threat of network surveillance, so we do not use any extra infrastructure.

We summarize the above in Table 1. Our defenses share the category of zero-delay lightweight defenses with only WTF-PAD. Noting that WTF-PAD is defeated by DF [20], we compare our work with theirs to show that our defenses are effective against DF.

3 Preliminaries

3.1 Threat Model

Like previous works in WF, we consider a passive adversary who is local to the user. Figure 1 illustrates the attack model. The adversary sits between the user and the entry node of the Tor network, eavesdropping on the network traffic over the encrypted channel. The adversary will not delay, modify or drop any packets.

We aim to deploy our defenses on Tor nodes to protect its clients against WF. There are three nodes in a typical Tor circuit: entry, middle, and exit. The middle node would serve as the cooperating proxy enacting the defense with the client. We use the middle node because the entry node is a possible WF attacker. Exit nodes and web servers would be entirely unaffected by our defense, as the middle node would drop dummy packets.

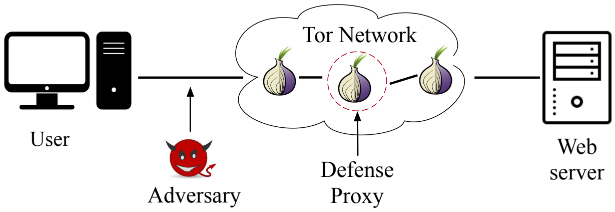

Figure 1: The threat model for WF. The adversary sits between the user and the Tor network. The middle node of Tor network will be a cooperating proxy to deploy our defense.

3.2 Classification

From the attacker's perspective, WF can be regarded as a classification problem. During webpage loading, a WF attacker records network traffic traces (also known as *packet sequences*). The attacker visits a certain set of monitored pages in advance and trains a machine learning model on these traces. Each webpage is a class, and a particular trace belonging to this class is called an *instance*. Then, when observing the client's traces, the attacker predicts which webpage the trace belongs to, based on the trained model.

WF attacks may be evaluated in either the *closed-world* or the *open-world* scenario. In the closed-world scenario, we assume a user only visits a specific set of webpages, also called *monitored webpages*. In the open-world scenario, the client can also visit non-monitored webpages, so the attacker must predict whether a trace is a monitored one or a non-monitored one. If it is monitored, the attacker has to further answer which one. The attacker never trains on the same webpage the client visits; therefore, the attacker has zero prior knowledge of the client's behavior. We focus on the more realistic open-world scenario.

While it is more difficult than the closed-world scenario, a large number of attacks have recently shown open-world success [7, 16, 17, 18, 24, 25, 30].

In the closed-world scenario, the attacker must achieve high accuracy (true positive rate), while in the open-world scenario, the attacker must achieve both high accuracy and precision. Therefore, to prove the efficacy of our defense against open-world attackers, we need to ensure that the attacker has both low accuracy and low precision. We specifically define precision in open-world WF below.

3.3 Precision

The precision of a classifier is defined as the proportion of positive (i.e. monitored) classifications that are correct. Researchers have pointed out that the base rate (the proportion of monitored webpages visited by the client) has been erroneously ignored in previous WF works when calculating precision [23]. This may lead to the base rate fallacy: an attack that seems to be accurate (high true positive rate) is actually highly imprecise when the base rate is low. If it is imprecise, its classifications are useless to the attacker. The definition of precision is somewhat atypical for WF because it is not a two-class problem, so we explicitly define it as follows:

Definition 3.1. Positives. If the WF attacker classifies a trace as belonging to a monitored webpage, it is a positive. If the classification is correct, it is a *true positive*. If the classification is incorrect and the sequence actually belongs to a different monitored webpage, it is a *wrong positive*. If the classification is incorrect and the sequence actually belongs to a non-monitored webpage, it is a *false positive*.

Definition 3.2. Precision. In an experiment, let N_P and N_N denote the number of positives and negatives respectively. Let TPR and WPR denote the proportion of true positives and wrong positives to N_P. Let FPR denote the proportion of false positives to N_N. Then the precision is:

$$\pi = \frac{TPR}{TPR + WPR + r \cdot FPR},$$

In the above, r is the ratio between how often the client visits non-monitored webpages to how often the client visits monitored webpages. A higher r lowers precision, and makes the open-world classification problem harder; previous attacks have shown success against clients up to $r = 1000$ [23]. We want to prove that our defense is effective even for low-r clients that visit monitored webpages frequently. Therefore, in our paper, we set $r = 10$, representing a client that visits one monitored webpage for every ten non-monitored webpages. Hereafter we evaluate precision for such a client.

We also present the F_1 score, the harmonic mean of TPR and precision, as a single combined metric for comparison between different attacks.

3.4 Overhead

We define the overhead of defending a trace as follows.

Definition 3.3. Trace. A trace is a sequence of packets collected during a page loading process, denoted as $P = \langle (t_1, L_1), (t_2, L_2), \cdots, (t_{|P|}, L_{|P|}) \rangle$ where $|P|$ is the total number of cells in the trace. t_i is the timestamp of the i-th packet. L_i shows the direction and length of the i-th packet. Tor uses its own datagrams called cells which are all padded to the same length. Since Tor cells are of the same length, we simply use $L_i = +1$ to represent a cell coming from the client and -1 to represent a cell coming from the server. (We use packets to refer to both types of datagram.)

Definition 3.4. ℓ-trace. An ℓ-trace comprises traces of consecutive visits to ℓ webpages, denoted as $P = P_1 || P_2 || \cdots || P_\ell$.

Let P denote the original trace and P' denote the trace after implementing some defense D. We define latency and data overhead on this trace as follows, which are the costs of implementing the defense D:

Definition 3.5. Latency overhead. The latency overhead $T(D)$ of defense D on P is the extra time taken to transmit real packets, divided by the original transmission time. Denote the last **real** packet in P' as t_k, then we have:

$$T(D) = \frac{t_k - t_{|P|}}{t_{|P|}}$$

Definition 3.6. Data overhead. The data overhead $O(D)$ of defense D on P is the total amount of dummy data divided by the total amount of real data:

$$O(D) = \frac{|P'| - |P|}{|P|}$$

Generally, latency overhead affects users' browsing experience while data overhead shows the extra burden laid on the network. They should be considered together when evaluating a defense. Following previous works [3, 11, 24, 27], we define these two metrics to be independent of each other, to simplify the analysis and to more easily highlight how defenses change each overhead. When bandwidth is a concern, for example, increasing the bandwidth overhead will likely delay page loading but will not change the time overhead.

Note that Definition 3.5 does not include the whole trace P', only the sequence up to the last real packet. That is because the client's page would have fully loaded upon reception of the last real packet; extra dummy packets sent or received after that point have no effect on the client's experience. Our defenses, FRONT and GLUE, have zero latency overhead (zero-delay) and little data overhead (lightweight).

4 FRONT

In this section, we first introduce the high-level idea behind FRONT by pointing out our observations and intuition in Section 4.1. In Section 4.2, we describe its design in detail. Finally, in Section 4.3, we analyze the features of our defense. We will evaluate FRONT in Section 5.

4.1 Overview

Learning from previous failures to implement WF defenses on anonymity technologies like Tor, we believe three properties are necessary to achieve deployability: zero-delay (no latency overhead), lightweight (small data overhead), and easy implementation. This respectively ensures that the defense has no effect on user experience, its extra data can be easily borne by the anonymity network, and its codebase will be easy to understand and maintain. Seeing the failure of confusion and regularization strategies to achieve these properties in previous work, we turn to obfuscation, and create FRONT (**F**ront **R**andomized **O**bfuscation of **N**etwork **T**raffic).

The only known defense that shares these properties with FRONT is WTF-PAD [11]. In WTF-PAD, the client and server separately maintain two histograms where they sample inter-arrival time to generate dummy packets. To achieve the best performance, they also suggest tuning the parameters by sampling inter-arrival time from the real dataset. However, the tuning process is not user friendly and the construction and maintenance of these histograms are nontrivial. Pulls [22] also points out that the token mechanism in WTF-PAD is unnecessary and should be abandoned.

By contrast, FRONT is much simpler, uses less data overhead, and achieves better performance against the best attacks. It relies on two **key intuitions**:

- **Obfuscating feature-rich trace fronts**. The first few seconds of each trace, which we call the trace front, leaks the most useful features for WF classification. Some of the best attacks explicitly use the trace front for classification [7, 24]. We dedicate most of our data budget to obfuscating the trace front, instead of spreading them evenly over the trace.

- **Trace-to-trace randomness**. FRONT adds dummy packets in a highly random manner, ensuring different traces of the same webpage look different to each other in total length, packet ordering, and packet directions. To do so, it randomizes the data budget and the region where we inject dummy packets. Since we must allow the attacker to train on defended traces instead of original traces, trace-to-trace randomness hurts the attacker's ability to find any meaningful patterns for a webpage class. Most regularization defenses suffer from trace-to-trace consistency.

Table 2: Defense parameters and variables in FRONT. Defense parameters set the overhead and behavior of FRONT, while trace variables are drawn from corresponding defense parameters for each trace separately to ensure trace-to-trace randomness.

	Notation	Parameter
Parameters	N_c	Client's padding budget
	N_s	Proxy's padding budget
	W_{min}	Minimum padding time
	W_{max}	Maximum padding time
Variables	$n_c \leftarrow \bar{U}(1, N_c)$	Number of outgoing dummy packets
	$n_s \leftarrow \bar{U}(1, N_s)$	Number of incoming dummy packets
	$w_c \leftarrow U(W_{min}, W_{max})$	Client's padding window
	$w_s \leftarrow U(W_{min}, W_{max})$	Proxy's padding window

4.2 Defense Design

There are three steps in using FRONT to defend a trace: sample a number of dummy packets, sample a padding window size and schedule dummy packets. Its parameters are summarized in Table 2.

Sample a number of dummy packets N_c and N_s are two parameters determining the data overhead of FRONT, respectively representing the client's padding budget and the proxy's padding budget. For each trace, the client samples n_c from the discretized uniform distribution between 1 and N_c, denoted as $\bar{U}(1, N_c)$; the proxy samples n_s from $\bar{U}(1, N_s)$. n_c and n_s are the actual number of dummy packets they will inject into that trace.

Sample a padding window FRONT spends most of its budget obfuscating trace fronts. To do so, both client and proxy will first generate a *padding window*, controlling where most dummy packets are expected to be injected into the original trace. For each trace, the client samples w_c from the uniform distribution between W_{min} and W_{max}, denoted as $U(W_{min}, W_{max})$; the proxy samples w_s from the same distribution. The reason we set a lower bound W_{min}, instead of 0, is to ensure that the generated padding window size is not too small; if it is too small, the defense may require an extreme bandwidth rate to support.

Schedule dummy packets After sampling the above variables, the client and proxy generate separate timetables to schedule when their respective n_c and n_s dummy packets will be sent. They generate the timestamps by sampling n_c and n_s times from a Rayleigh Distribution. Its probability density function is:

$$f(t; w) = \begin{cases} \frac{t}{w^2} e^{-t^2/2w^2} & t \geq 0 \\ 0 & t < 0 \end{cases},$$

where w is w_c for the client and w_s for the proxy. True packets will be sent with no delays and dummy packets will be

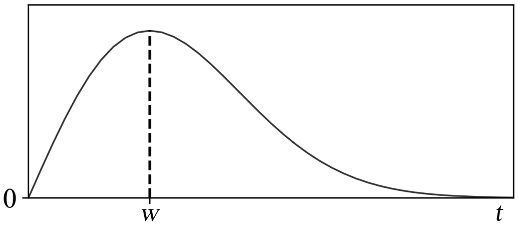

Figure 2: PDF of Rayleigh Distribution.

sent according to the timetables. When webpage loading finishes, the client will notify the relay with a packet and any unsent packets left in the timetable are simply dropped.

4.3 Defense Analysis

FRONT makes use of a Rayleigh Distribution. The corresponding PDF $f(t;w)$ is shown in Figure 2. The curve first increases quickly, peaks at w and then gradually decreases. This results in a burst of dummy packets at the start of a trace, in accordance with our first intuition. Though our dummy packet window has a nominal length of w, the window is "soft"; we expect 40% of the dummy packets to lie in the time interval [0, w]:

$$\int_0^w \frac{t}{w^2} e^{-t^2/2w^2} \mathrm{d}t \approx 0.40$$

We sample the number of dummy packets and padding window size so that they are different each time we load a webpage, even if it is the same webpage. This eliminates possible patterns that could be leveraged by an attacker, as suggested by our second intuition.

In FRONT, the latency overhead is always 0 since it never delays any real packets whereas the data overhead is proportional to $N_c + N_s$. The number of dummy packets in each trace will be $\bar{U}(1, N_s) + \bar{U}(1, N_c)$ (unless they are cut off by the end of a real trace), with a mean of $(N_s + N_c)/2 + 1$.

5 Evaluation of FRONT

In this section, we evaluate FRONT in several aspects. After presenting our experimental setup, we evaluate FRONT against the best attacks to show that it is able to defeat them, and do so more efficiently than the state-of-the-art defenses. We follow up with an analysis of our design decisions to show why FRONT succeeds.

5.1 Experimental Setup

To conduct our experiments, we collect a new dataset (denoted as `DS-19`) between February and April 2019 with Tor Browser 8.5a7 on Tor 0.4.0.1-alpha, driven by command-line calls to Tor Browser. We visited the homepages of Alexa top 100 websites 100 times each as our monitored webpages and 10000 other webpages as our non-monitored webpages, filtering out pages that did not load (such as those inaccessible through Tor). In doing so, we used a single machine connected to a university network. Since all traces are collected from an automated browser and none of them are from real users, there are no ethical concerns regarding the dataset and the following experiments.

Table 3: Defense settings and corresponding overheads.

Defense	Parameters	Overhead (%)	
		Latency	Data
No defense	-	0	0
Tamaraw [3]	$\rho_{out} = 0.04, \rho_{in} = 0.012, L = 50$	78.43	162.93
WTF-PAD [11]	Normal rcv	0	32.71
FT-1	$N_s = N_c = 1700, W_{min} = 1\,\mathrm{s}, W_{max} = 14\,\mathrm{s}$	**0**	**33.01**
FT-2	$N_s = N_c = 2500, W_{min} = 1\,\mathrm{s}, W_{max} = 14\,\mathrm{s}$	**0**	**48.80**

We choose two defenses, WTF-PAD [11] and Tamaraw [3], as competitors to our defense representing two extremes in design philosophy: WTF-PAD is a lightweight obfuscation defense, while Tamaraw is a heavyweight regularization defense with high latency and data overhead. Other obfuscation defenses have been broken by known attacks, while most confusion and regularization defenses are either more expensive than Tamaraw, or impractical to implement.

We use kNN [24], CUMUL [16], kFP [7] and DF [20] as benchmarks to evaluate the defenses. We use suggested parameters in their papers for kNN, kFP and DF with one exception: for DF, we set the maximum length of the traces to 10000 (instead of 5000 suggested by Sirinam et al. [20]) to accommodate our dummy packets. CUMUL uses an SVM, which is heavily dependent on choosing the correct parameters, so we first follow the paper to perform parameter tuning on the candidate parameters and find the optimal parameters.

All the experiments are conducted in open-world setting. For each attack, we apply 10-fold cross validation on the dataset. We count true positives, wrong positives and false positives on each fold and add them up together. Then we calculate their corresponding TPR, WPR, FPR and precision.

5.2 Evaluation against Other Defenses

We start by showing that FRONT dominates WTF-PAD in terms of effectiveness against the best attacks. We also evaluate Tamaraw, a heavyweight defense, for comparison. The overhead of each defense is shown in Table 3. We choose two sets of parameters for FRONT: FT-1 representing a lightweight defense which has similar data overhead as WTF-PAD and FT-2 representing a defense with slightly higher overhead but greater effectiveness. $N_s + N_c$ determines the data overhead while W_{min} and W_{max} decide the padding window size. We put a more detailed discussion about how to set these parameters in Appendix A.

Table 4: Defense performances on DS-19. A lower F_1 score represents a better defense.

Defense	TPR (%)				Precision (%)				F_1			
	kNN	CUMUL	kFP	DF	kNN	CUMUL	kFP	DF	kNN	CUMUL	kFP	DF
No defense	89.09	94.44	91.85	96.40	83.18	64.22	94.38	91.12	0.86	0.76	0.93	0.94
Tamaraw [3]	3.41	3.85	2.08	0.58	2.33	8.13	23.16	6.78	0.028	0.052	0.038	0.11
WTF-PAD [11]	9.35	55.55	52.97	81.99	51.52	18.53	70.69	60.92	0.16	0.28	0.61	0.70
FT-1 (This work)	**2.56**	**36.08**	**43.03**	**70.82**	**41.22**	**11.97**	**71.19**	**34.88**	**0.048**	**0.18**	**0.54**	**0.47**
FT-2 (This work)	**0.83**	**26.19**	**34.31**	**58.95**	**37.22**	**8.52**	**68.33**	**30.59**	**0.016**	**0.13**	**0.46**	**0.40**

FRONT performance on DS-19 Table 4 shows how well WF attacks perform against our evaluated defenses. We present TPR, precision and F_1 score of each attack under different defenses.

When no defense is implemented, all attacks achieve over 89% TPR. kFP and DF become the strongest attacks since F_1 is over 90% for both of them. Even though CUMUL's TPR is quite high (94%), it has the lowest precision (64%), resulting in its low F_1. All attacks achieve a low F_1 score against Tamaraw, but Tamaraw comes with a very high price in terms of overhead. WTF-PAD is much cheaper at 32% data overhead, and it defends against kNN and CUMUL well. However, kFP and DF remain effective against WTF-PAD, achieving 0.61 and 0.70 F_1 score.

FT-1 outperforms WTF-PAD in defending against every attack, especially kNN and DF. With nearly the same data overhead as WTF-PAD, kNN performs just as poorly against FT-1 as against Tamaraw, and DF performs half as well as before (by F_1 score). FT-2 further decreases F_1 of the strongest attacks, kFP and DF, with only 48% data overhead.

We find that FRONT is especially effective against kNN, even approaching Tamaraw's performance. It is also effective at defeating DF, the strongest attack on the undefended dataset: the precision of DF drops significantly against FRONT, more so than kFP.

To explain why FRONT outperforms WTF-PAD, we evaluate how it embodies our two key intuitions (in Section 4.1) compared to WTF-PAD. First, to show its obfuscation of trace fronts, we calculate how much data budget FRONT and WTF-PAD use in each portion of the trace. WTF-PAD distributes its budget evenly: it spends 24% of its budget in the first quarter of the trace and 49% in the first half. In contrast, FRONT uses 40% of its budget in the first quarter and 69% in the first half. Second, to show trace-to-trace randomness, we evaluate the coefficient of variation of dummy packets injected in each webpage class. We find that FRONT has a median coefficient of variation of 42% compared to 36% for WTF-PAD over our dataset.

TPR on different websites We further investigate FRONT's webpage-to-webpage performance on DS-19. We equally divide the monitored webpages into 4 groups based on their webpage sizes, denoted as G_1, G_2, G_3 and G_4, where G_1 is the smallest quartile of webpages and G_4 is the largest quartile of webpages. The number of packets of each webpage in those groups is up to 2039, 4368, 6611 and 28199, respectively. We can see that the sizes of webpages vary greatly, especially for G_4.

We choose the most precise attack, kFP, and compute the recall. The recall on each group is 24%, 24%, 35% and 54%. The performance of FRONT does not change much on first three groups. The webpages in G_4 are 10 times larger than G_1, and the recall rate increases by 30%.

FRONT performance on DS-14 We did a supplementary experiment on Wang's dataset [24] collected in 2014 (denoted as DS-14) which consists of 9000 monitored webpages and 9000 non-monitored ones. The mean number of packets is 2163 in DS-14 and 4444 in DS-19. Therefore, the web pages of DS-14 are significantly smaller. The intent of this experiment is to verify that FRONT works on different websites. With 41% data overhead, FRONT greatly outperforms over WTF-PAD (which has 44% data overhead) in all metrics, no matter which attack is used.

For the strongest two attacks, kFP and DF, FRONT reduces their F_1 score to 0.30 and 0.41, compared with 0.48 and 0.63 against WTF-PAD. The most significant case is kNN which relies greatly on trace FRONT information. Its F_1 is reduced to only 0.03 while WTF-PAD reduces it to 0.26. We also find that all the attacks perform better on DS-19 than DS-14. The observation that larger websites are easier to identify was also made by Overdorf et al. [15].

5.3 Information Leakage Analysis

Some recent works [5, 12] have pointed out that empirically evaluating a defense against state-of-art attacks may not show the real security level of such a defense. WeFDE, proposed by [12], quantifies the amount of information leakage for 3043 features, chosen from those exploited by known state-of-the-art attacks. We use WeFDE to measure the information leaked on undefended traces, WTF-PAD and FRONT. The detailed methodology of WeFDE and the introduction of feature set can be found in [12].

We plot the empirical cumulative distribution function (ECDF) of information leakage for all features in Figure 3. Generally speaking, the curve for FRONT increases much

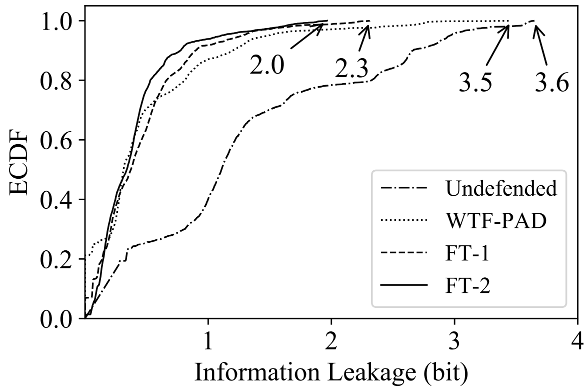

Figure 3: ECDF of information leakage on different datasets. The 100th percentile points are marked.

faster than that for WTF-PAD and undefended Tor, indicating that most features leak less information under FRONT. Specifically, no feature leaks more than 3.6 bits of information on undefended Tor, 3.5 bits for WTF-PAD, 2.3 bits for FT-1 and 2 bits for FT-2.

The information leakage analysis confirms again that FRONT achieves a higher security level than WTF-PAD. We include the full information leakage result in Appendix D.

5.4 Choosing Where to Pad

FRONT is built on the intuition that it helps to obfuscate the trace front for defense effectiveness. We validate this intuition here by delaying all dummy packets from 0 s to 10 s and measure the change in TPR and precision of each attack. We set W_{min} = 1 s, W_{max} = 14 s, $N_c = 1000$ and $N_s = 1000$ based on FT-2. Figure 4 shows the results. The larger the delay, the less we will obfuscate the trace front (the same padding budget is instead spent on the middle or the end).

We can see that for all attacks, both TPR and precision grow as we increase the delay, thus leaking more of the trace front. All attacks' TPR increase by 5–30%, among which DF always achieves the highest TPR, increasing from 59% to 71%. kFP's TPR nearly doubles from 34% to 62%. In terms of precision, there are some ups and downs due to its sensitivity to false positives. But still, all attacks become 6–15% more precise as we reveal the trace front by shifting dummy packets later. This experiment indicates that trace fronts do leak a lot of information.

5.5 Impact of Randomness

In FRONT, the client and proxy will sample the number of dummy packets and the padding window from a range instead of fixing them. We do two experiments to validate the effectiveness of this design decision.

In the first experiment, we gradually shrink the range of choices for the number of dummy packets and observe the change of TPR and precision for each attack. As before, we have $W_{min} = 1$ s, $W_{max} = 14$ s, $N_c = N_s = 2500$ based on FT-2. Unlike previous experiments, here we sample n_c from $\bar{U}(\beta \cdot N_c, N_c)$ and n_s from $\bar{U}(\beta \cdot N_s, N_s)$. We vary β, which controls the degree of randomization, from 0 to 1 (0 being maximal randomization).

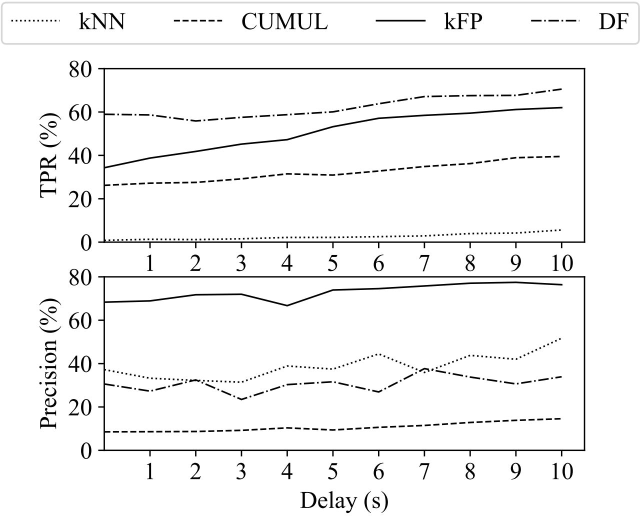

Figure 4: Change in performance of WF attacks when all dummy packets are delayed by 1 to 10 seconds.

As β increases, trace-to-trace randomness in the number of dummy packets decreases. Figure 5 shows the results. We see that increasing β weakens the defense, as all attacks except DF increase in TPR, especially when $\beta > 0.8$. As for DF, its TPR remains around 60%: increasing trace-to-trace randomness does not weaken its attack ability. Note that increasing β also increases data overhead linearly, doubling the data overhead at $\beta = 1$.

We perform a similar experiment on randomizing padding window size, using the same settings as the above experiment. We keep $W_{max} = 14$ s and set $W_{min} = \beta \cdot W_{max}$. We gradually increase β from 0 to 1. Figure 6 shows the results. Just as before, when we decrease the randomness in padding window size, TPR increases, especially for CUMUL and DF.

5.6 Evaluation of Data Overhead

In this part, we want to measure how an increase in the overhead budget affects the attacker's effectiveness. We focus on kFP here because the extensive experiments in Section 5.2 to Section 5.5 show us that kFP is the strongest attack by F_1 score; DF is accurate but imprecise against FRONT, so its F_1 score is lower. Setting $W_{min} = 1$ s and $W_{max} = 14$ s, we vary $N_s + N_c$ from 0 to 7200 packets in intervals of 200 packets. We show TPR and F_1 of kFP in Figure 7.

Without FRONT, kFP can achieve 92% TPR. Its TPR de-

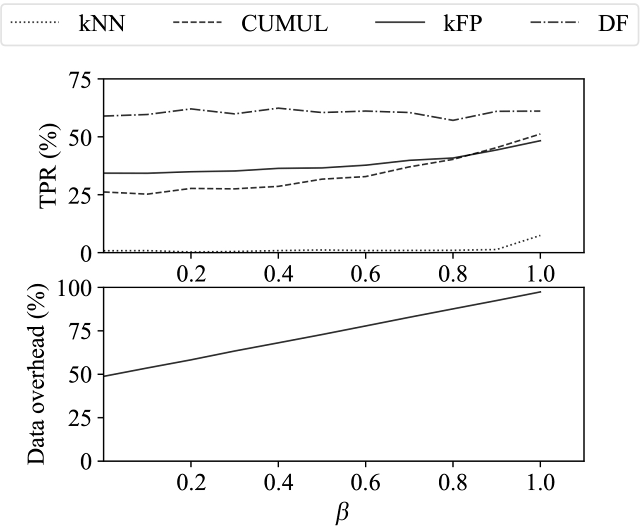

Figure 5: TPR and data overhead while varying β to change the lower bound of padding budget.

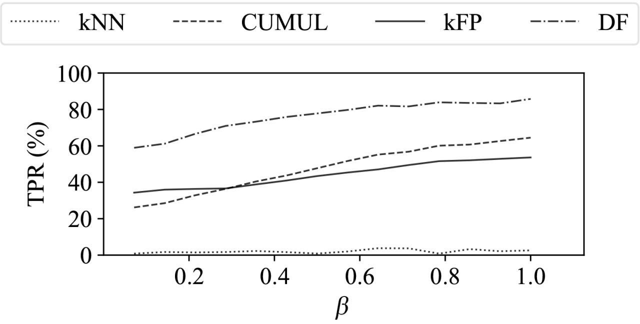

Figure 6: TPR while varying β to change the lower bound of padding window size. Data overhead remains constant.

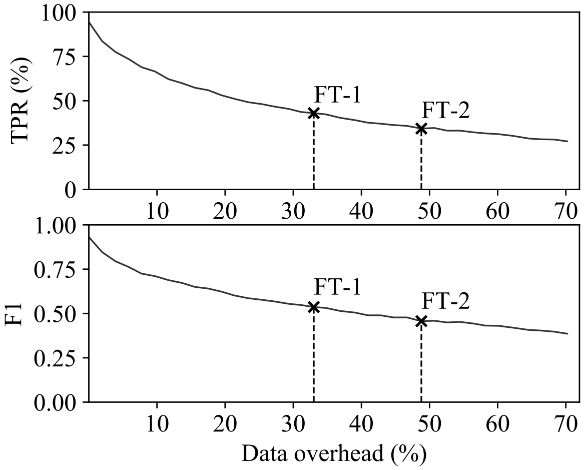

Figure 7: kFP's TPR and precision on protected traces given different data overhead budgets. We mark with a cross the data overheads of FT-1 and FT-2 on the figure.

creases quickly as we initially increase the size overhead. With only 25% data overhead, its TPR is already lower than 50%. On the other hand, its F_1 score decreases from 93% to 38% as we increase the data overhead from 0% to 70%.

6 GLUE

Our second proposed defense, GLUE, exploits an entirely new facet of website fingerprinting to achieve even greater success than FRONT against known attacks. We start by presenting the big picture of what GLUE exploits and how it achieves success. Due to the novelty of GLUE, we carefully elaborate our threat model and defense design.

6.1 Overview

Many WF attacks have been published in recent years, all of them relying on the same assumption: that every trace the attacker must classify corresponds to exactly one webpage. We call these traces *singleton traces*. This is satisfied if the client dwells on pages for some time before visiting the next page; the attacker will notice the obvious time gap and *split* the trace at that point. Even a second of inactivity will be enough. Conversely, when the client visits $\ell \geq 2$ pages consecutively without an obvious time gap — for example, by clicking a link before the page has fully loaded — all known WF attacks cannot succeed in classifying the ℓ-trace thus generated, even if they are properly trained and aware of such a possibility [10, 30].

Since known WF attacks can only classify singleton traces ($\ell = 1$), there are two difficult problems the attacker must solve to classify ℓ-traces correctly for $\ell \geq 2$. First, the attacker must correctly determine ℓ; we call this the **split decision problem**. Secondly, the attacker must find $\ell - 1$ points to split the ℓ-trace into ℓ separate singleton traces; we call this the **split finding problem**. Then, the classifier can input these singleton traces into a powerful WF attack. There are some works suggesting that the latter problem could be solved for $\ell = 2$ [26, 29], but no solution is known in general; the former problem has never been solved.

We leverage the difficulty of solving these problems to create a new defense, GLUE. Whenever the client is dwelling on a webpage, GLUE adds dummy packets to make it seem as if the client is visiting new pages consecutively. GLUE will stop sending dummy packets when the client loads a new page, thus hiding the true start of the next page. In other words, GLUE tries to glue together singleton traces into ℓ-traces for large values of ℓ. Unable to solve either the split decision or finding problem, attacks are very likely to fail if they split traces wrongly. This is especially true if the resultant singleton traces have extra packets in the trace front, which is critical for correct classification.

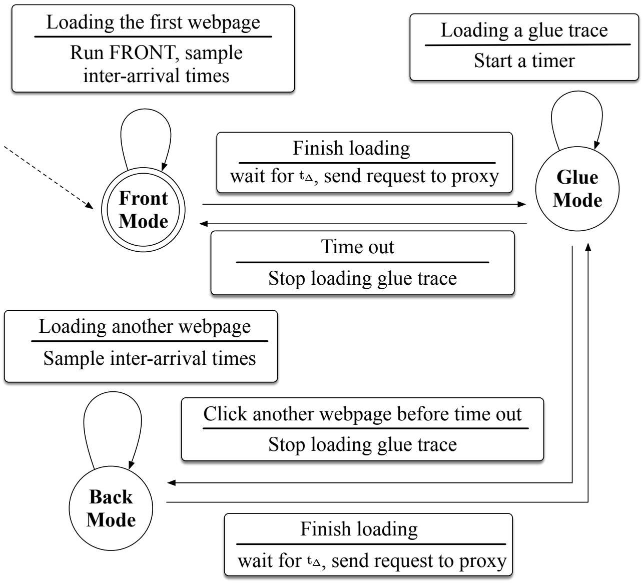
Loading the first webpage
Run FRONT, sample inter-arrival times
Loading a glue trace
Start a timer
Finish loading
wait for t_Δ, send request to proxy
Front Mode
Glue Mode
Time out
Stop loading glue trace
Loading another webpage
Sample inter-arrival times
Click another webpage before time out
Stop loading glue trace
Back Mode
Finish loading
wait for t_Δ, send request to proxy

Figure 8: Client's state machine. Starting from Front Mode, it will switch between Glue Mode and Back Mode until dwell time becomes too long. "Glue traces" are padded in Glue Mode. The client keeps sampling inter-arrival times in both Front and Back Mode.

6.2 Defense Design

Suppose a client visits ℓ webpages in a time period and then stops. GLUE tries to make sure that the attacker will see a seemingly consecutive ℓ-trace $P = P_1||P_2||\cdots||P_\ell$. Without GLUE, they may have dwell time gaps between them, allowing the attacker to split them trivially.

Denote the dwell time on P_i as d_i. GLUE pads for a maximum duration d_{max}. For GLUE to create an ℓ-trace, let us suppose $d_i \leq d_{max}$ for $i = 1, \ldots, \ell - 1$ and $d_\ell > d_{max}$. While the client dwells on webpages, the client and the proxy will send each other dummy packets. Figure 8 gives the state machine of a client, and the proxy's state machine is similar. GLUE also uses FRONT noise to defend the first of ℓ-traces.

Front Mode Starting in Front Mode, our defense waits for the client to visit a webpage. When the client does so, we will add dummy packets according to our FRONT defense, as described in previous sections. We will also sample those inter-arrival times between incoming packets and outgoing packets to obtain some distribution I. After the client finishes visiting the webpage, we sample t_Δ according to I (described below), wait for time t_Δ, then switch to Glue Mode.

Glue Mode In Glue Mode, the client and proxy send each other dummy packets in such a way that it looks as if the client decided to visit a new, random webpage. (The person behind the client is actually dwelling on the previous webpage.) They will do so for at most time d_{max}. They immediately stop doing so if the client actually decides to visit a webpage before d_{max} has passed: the client will notify the proxy to terminate Glue Mode as well. If the client dwells on the webpage for longer than d_{max}, the algorithm will consider the client inactive and return to Front Mode. Otherwise, it will go to Back Mode. We call the dummy packets added here "glue traces".

Back Mode In Back Mode, the client is visiting another webpage. This is like Front Mode, except we add zero dummy packets. We still sample packet inter-arrival times and switch back to Glue Mode after waiting for t_Δ.

GLUE incorporates FRONT in Front Mode, ensuring that the first trace of any ℓ-trace will be padded with FRONT. This is because we found that GLUE alone does not protect the first trace well (shown in Section 7.6), but achieves excellent protection of all other traces. We need to add a bit of overhead to protect the first trace.

In the above, I is the inter-arrival time distribution only with those time gaps between an incoming and an outgoing packet. t_Δ is the sampled inter arrival time. We choose $t_\Delta \in U(I_{\bar{20}}, I_{\bar{80}})$ where $I_{\bar{20}}$ and $I_{\bar{80}}$ are the 20 percentile and 80 percentile of the inter-arrival time distribution I respectively. We intentionally create such a small gap to simulate a time interval when the client sends out some request after receiving data from the server during a webpage loading. By doing so we connect real traces with glue traces together naturally without any abnormal gaps in between. We also randomize d_{max} by sampling from a uniform distribution so that the attacker could not trivially remove the noise on the tail.

We illustrate how GLUE works with Figure 9. Suppose a client visits three webpages with real traces P_1, P_2, and P_3, then stops, with time gaps $d_1, d_2 < d_{max}$ after the first two pages respectively. The attacker will collect a 3-trace, $P' = P'_1||P'_2||P'_3$. P'_1 contains P_1 with FRONT noise, followed by a glue trace of duration d_1. P'_2 contains P_2 followed by a glue trace of duration d_2. P'_3 contains P_3 followed by a glue trace of duration d_{max}. Of course, the attacker cannot know where each trace starts or ends. In fact, the attacker will not even know how many traces there are. If the attacker tries to split the combined trace incorrectly, some or all of the split traces will be contaminated by dummy packets in their beginning or end, which greatly affects WF attack performance.

6.3 Distributing Glue Traces

To make sure glue traces look like real traces, the client needs to have a database which contains real webpage loads. We propose that the client will retrieve such a database along with the list of Tor nodes at Tor startup from Tor directory servers. Then the client will ask for more after some certain period of time. During Glue Mode, the client instructs the proxy when to send a dummy packet.

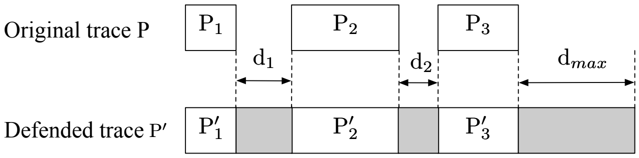

Figure 9: A toy example of what traffic looks like with GLUE. The white boxes are real traces while the grey boxes are glue traces (made with dummy packets). Glue traces remove time gaps between real traces to exploit the difficulty of the split decision and finding problems.

Note that glue traces contain no real data, only *timestamps* of when dummy packets are sent and received. Therefore, we do not expect the traces to cause much extra data overhead. We estimate the data overhead for distributing glue traces as follows. On average a trace has 4441 packets in our dataset. Therefore, the average web page size is 2.3 MB. Suppose a timestamp takes up 2 bytes, then one glue trace takes up $4441 \times 2 = 0.008$ MB. Hence, on client side, the data overhead in the long run will be $0.008/2.3 \approx 0.003$ if the number of glue traces downloaded is the same as the number of web pages visited; and it will be 0.03 if the client downloads 10 times more glue traces than actually needed ones.

On the directory server side, we estimate the distribution cost as follows. Taking statistics from November 2018 to November 2019[1], we found that the average bandwidth spent on answering directory requests is 172 MB/s and the average number of Tor users is 2.1 million per day. If the average user downloads 200 glue traces per day, the average bandwidth for distributing glue traces is about 39 MB/s. Therefore, the directory server is expected to have about $39/172 \approx 23\%$ data overhead. To obfuscate user activity, we can require users to download a randomized number of glue traces regularly even if they do not need to, using padding to hide the number of glue traces downloaded from an eavesdropper.

6.4 Solving Split Decision and Split Finding

To break down an ℓ-trace, the attacker pursues the following strategy: determine ℓ (split decision problem) and then find $\ell - 1$ points to split the ℓ-trace (split finding problem). To the best of our knowledge, there is no prior work on split decision, and only two studies looking into split finding for 2-traces. Wang and Goldberg [26] put forward a split finding algorithm using kNN with a score system. Xu et al. [29] suggested using XGBoost to output the outgoing packet with the highest probability to be the split. They show that their algorithm could achieve better performance than kNN. Neither work considers ℓ-traces for $\ell \geq 3$.

Since GLUE relies on these problems being difficult, we want to make a sincere best effort at solving both problems for general ℓ so that future work will not be likely to break GLUE. To do so, we put forward a new framework: Coarse-Decided Score-Based (CDSB). CDSB performs better than both previous algorithms for any general ℓ.

Split decision We use a Random Forest classifier with 511 features extracted by expert knowledge to decide how many splits there are. Intuitively, the more webpages we visit, the longer an ℓ-trace's transmission time will be. Since splits are time-sensitive, we include rich time information in our feature set. We also exploit volume information such as the number of packets and the number of outgoing packets in our feature set. Refer to Appendix B for a detailed feature list.

Split finding We extend Xu et al.'s XGBoost to score each outgoing packet in the trace; a higher-scoring packet is more likely to be the true split. However, the algorithm does not simply choose all the highest-scoring packets. Because usually all the packets around a true split score highly, but only one of them is the true split. If we chose all of them, we would have many false positives. Instead, we choose the highest-scoring packet as a split in each round, and we eliminate *nearby* packets from consideration as splits for future rounds. By this score decoding processing, we generate predicted splits. We put the pseudocode in Appendix C.

7 Evaluation of GLUE

In this section, we evaluate the performance of GLUE. We first present the experimental setup. GLUE creates ℓ-traces, but the exact value of ℓ is determined by client behavior; therefore, we evaluate GLUE on a range of values of ℓ, from 2 to 16. Finally, we investigate the overhead of GLUE, which is dependent on how long clients dwell on webpages.

7.1 Experimental Setup

We use `DS-19` to evaluate GLUE. We divide it into three parts: AttackTrain (9000 instances), SplitTrain (2000 instances) and Evaluation (9000 instances). We use AttackTrain to generate training data for WF attacks and split decision; SplitTrain to generate training data for split finding; and Evaluation to generate test data.

Due to its novelty, GLUE requires a new methodology to evaluate. The split decision problem is entirely unexplored, and although we make a good-faith attempt to solve it, we want to show that GLUE is still effective even if the attacker "cheats" by being given ℓ directly. While ℓ-traces are much harder to classify for large ℓ, the exact value of ℓ is dependent on user behavior. We want to show GLUE is powerful even for the minimal $\ell = 2$.[2] The split finding problem has been

[1] `https://metrics.torproject.org/`

[2] For $\ell = 1$, GLUE simply reduces to FRONT with some extra dummy packets at the end.

explored more and GLUE relies on its difficulty. Therefore, our evaluation is divided into two cases:

- ℓ-traces without split decision. We evaluate for $\ell \in [2,16]$, and the attacker is told the value of ℓ.
- ℓ-traces with split decision. We evaluate for $\ell \in [2,16]$, and the attacker must find ℓ.

7.2 ℓ-traces without Split Decision

We start with an investigation of ℓ-traces without split decision: the client visits ℓ pages with a moderately short dwell time between them. We tell the attacker what ℓ is, allowing the attacker to cheat by skipping the split decision problem. We use a lightweight setting for FRONT noise ($N_s = N_c = 1100$). We sample d_{max}, the maximum duration of glue traces, from $U(10\,\text{s}, 15\,\text{s})$. We assume the client's dwell time between webpages is a uniform distribution between 1 s and 10 s. The client visits ten times more non-monitored webpages than monitored webpages.

We randomly generate 4000 split points and 4000 non-split points from SPLITTRAIN as split training data. We randomly generate $\lfloor 9900/\ell \rfloor$ ℓ-traces for $\ell \in [2,16]$ from EVALUATION as test data so that we have 900 monitored webpages and 9000 non-monitored webpages in each test dataset.

The attacker will find $\ell - 1$ split points, split a ℓ-trace into ℓ singleton traces, and use a standard WF attack on each trace. Alternatively, the attacker could also find $2(\ell-1)$ splits and discard all packets between all odd and even splits, thus removing glue traces; however, we found that this strategy performs extremely poorly since it forces the attacker to find more traces accurately, so we do not present this strategy. Note that since the first singleton trace has some FRONT noise, the attacker should train two WF models: one "noisy model" trained on traces with FRONT noise to classify the first singleton trace; the other "clean model" trained without FRONT noise to classify the other singleton traces.

Figure 10 shows the performance of WF attacks after implementing GLUE. Increasing ℓ decreases both TPR and precision. In terms of TPR, DF performs the best at first (54% TPR at $\ell = 2$), but when more and more traces are glued together, it weakens quickly. When $\ell = 16$, all attacks achieve less than 5% TPR. We can see that despite being told ℓ and using our improved split finding procedure, the best WF attacks still cannot defeat GLUE.

7.3 ℓ-traces with Split Decision

In this experiment we tackle a more realistic scenario: the attacker does not know how many splits are in an ℓ-trace and thus needs to do split decision first. The client and the datasets are the same as in Section 7.2. We also generate 9000 ℓ-traces for $\ell \in [2,16]$ using ATTACKTRAIN to train for the split decision problem.

To evaluate the performance of WF attacks correctly, if the attacker guesses more than ℓ times for an ℓ-trace (due to incorrect split decision), we discard all the extra guesses and use only the first ℓ guesses.

Figure 11 shows the results. We can see that with split decision, WF attacks perform even worse, and their performance decreases more drastically with larger ℓ. When ℓ increases to 16, all WF attacks have less than 1% precision.

7.4 Undefended ℓ-traces

To show how attackers' performance are degraded by GLUE, we also test attack performance on the undefended dataset. We find that the best WF attack is kFP and it achieves 96% TPR at $\ell = 2$ down to 82% TPR at $\ell = 16$. It achieves 97% precision at $\ell = 2$ and 82% at $\ell = 16$. Split finding procedure has nearly no effect on kFP when ℓ is small and only a slight effect when ℓ is large. This is due to the high accuracy ($> 92\%$) of our split finding algorithm. Even if split decision is required, kFP still achieves 45%–75% TPR and 41%–77% precision. We put the detailed experiment results in Appendix E.

7.5 Analysis of Data Overhead

GLUE's data overhead consists of three parts: O_F incurred by FRONT noise, O_G incurred by glue traces except the last one and O_L incurred by the tail, i.e. the last glue trace. To estimate GLUE's data overhead, let the mean time taken to load a webpage be d_P. We take the average over user dwell times that are short enough to be glued, and denote it as d_G. The mean time of the tail is $d_L > d_G$. (d_L is the mean of the distribution from which we sample d_{max}.) For simplicity, we can assume that real and glue traffic have the same uniform packet rate b. Then,

$$\begin{aligned} O(\text{GLUE}) &= O_F + O_G + O_L \\ &= \frac{1}{\ell b d_P}\left[\frac{N_s + N_c + 2}{2} + (\ell-1) b d_G + b d_L\right] \\ &= \underbrace{\frac{N_s + N_c + 2}{2\ell b d_P}}_{\text{FRONT noise}} + \underbrace{\frac{(\ell-1) d_G}{\ell d_P}}_{\text{Glue trace}} + \underbrace{\frac{d_L}{\ell d_P}}_{\text{Tail}}. \end{aligned}$$

We can see that the $O(\text{GLUE})$ increases with users' dwell time and the duration of the tail while it decreases with ℓ, the number of pages glued together. Note that we only add FRONT noise for the first trace and the cost for that is shared by all the traces in an ℓ-trace, thus O_F is inversely proportion to ℓ. This is also the case for O_L. Since ℓ has little impact on O_G when it is large, we can reduce GLUE's data overhead with a large ℓ.

With $N_s = 1100$, $N_c = 1100$, the FRONT noise in our dataset has a mean of 24%. We also calculate the mean time

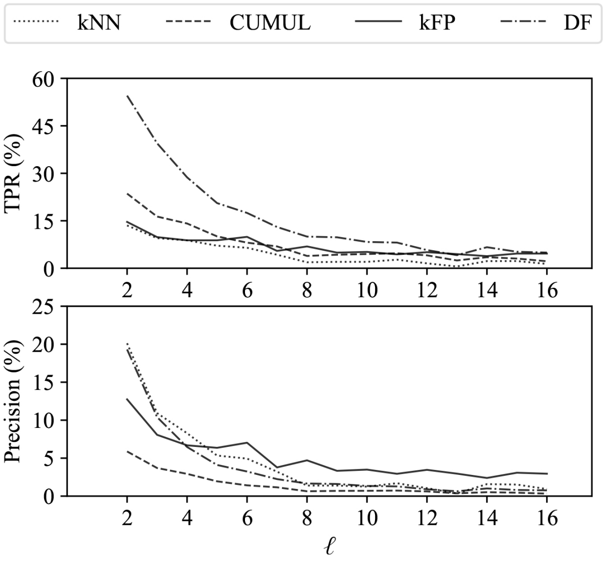

Figure 10: WF attack performance without split decision against GLUE on ℓ-traces.

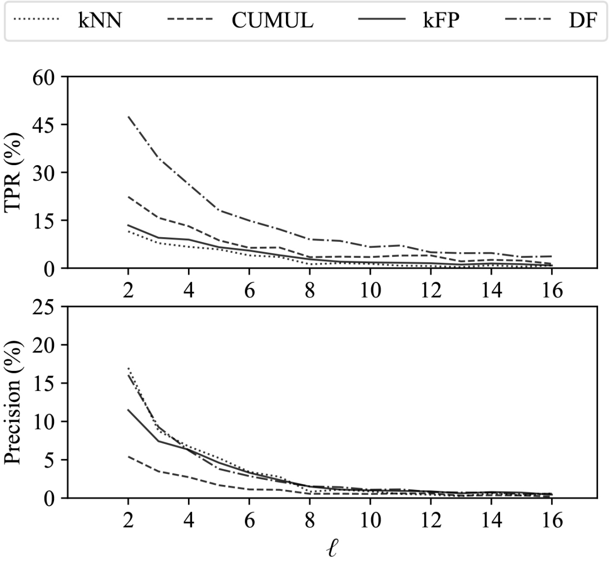

Figure 11: WF attack performance with split decision against GLUE on ℓ-traces.

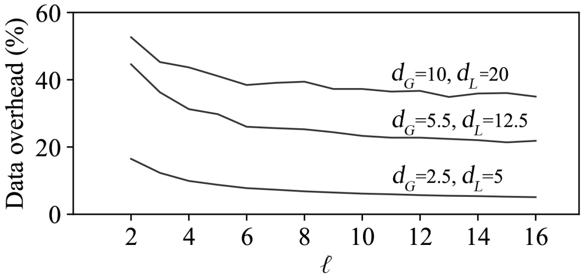

Figure 12: Data overhead with respect to different ℓ, d_G and d_L. Data overhead increases when d_G and d_L are larger.

to load a page based on our dataset and get $d_P = 27.30\,\text{s}$. Thus, the data overhead of GLUE is

$$O(\text{GLUE}) = \frac{0.24}{\ell} + \frac{\ell - 1}{27.30\ell} \cdot d_G + \frac{1}{27.30\ell} \cdot d_L. \quad (1)$$

We use three different settings to represent different client behaviors: $d_G = 2.5\,\text{s}$, $d_L = 5\,\text{s}$ as a strict version of GLUE, $d_G = 5.5\,\text{s}$, $d_L = 12.5\,\text{s}$ as normal GLUE and $d_G = 10\,\text{s}, d_L = 20\,\text{s}$ as lenient GLUE.

We apply these settings to our real datasets and show the results in Figure 12. The data overhead is 3% to 13% for strict GLUE, 22% to 44% for the normal GLUE, and 35% to 53% for lenient GLUE. The actual value within this range is dependent on ℓ, where larger ℓ reduces the overhead; we cannot determine ℓ because it depends entirely on client behavior. The values we found in Figure 12 are about 5–10% lower than equation (1) because most glue traces have uneven bandwidth density in reality.

7.6 Impact of FRONT Noise

In our defense design, we introduced some FRONT noise in the beginning. We evaluate FRONT noise separately here to show how it helps GLUE.

We use the same experiment setting as in Section 7.2 (WF attack on 2-traces to 16-traces without split decision), except that this time we do not add FRONT noise. We calculate TPR for only the first traces of ℓ-traces, and plot the results in Figure 13. Where there is no FRONT noise, all attacks could achieve 40%–80% TPR on the first traces; with little FRONT noise added, their TPR drops to 20%–60%.

Our observation is consistent with our discussion in Section 4. Even if the split for the first webpage is wrongly determined, the front portion is clean, leaking useful information to the attacker. Thus, it is necessary to protect the trace front.

8 Conclusion and Future Work

In this paper, we present two novel zero-delay lightweight defenses that are effective against the best WF attacks and easy to deploy on anonymity networks like Tor.

The first defense, FRONT, utilizes highly random noise to obfuscate traces. Instead of spreading dummy packets evenly, it focuses on obfuscating trace fronts. We also randomly sample the number of dummy packets and the packet padding window to ensure trace-to-trace randomness. With similar data overhead, it beats the best known lightweight defense, WTF-PAD, using a much simpler scheme.

We tested FRONT on two datasets collected five years apart, and on subsets defined by page sizes, and found that

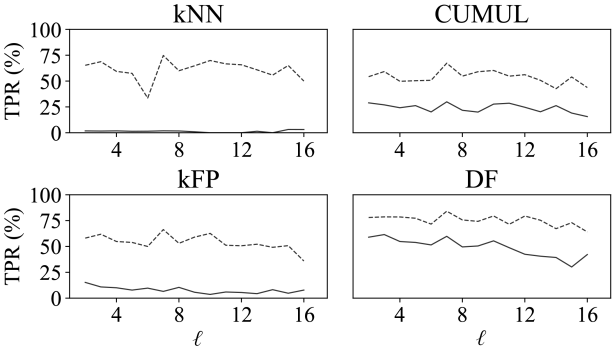

Figure 13: TPR on classifying the first page of ℓ-traces before and after adding FRONT noise. We use broken lines to show the result with no FRONT noise and full lines with FRONT noise.

FRONT's performance was generally not sensitive to either condition except that it performed worse on very large web pages. It could be true, however, that FRONT's performance may be affected if the client has poor network conditions (such that their own network serves as a bottleneck compared to Tor). We did not explore this situation; making FRONT automatically self-adjusting to poor network conditions is a potential future direction in this work.

The second defense, GLUE, forces WF attacks to confront two difficult problems, split decision and split finding, by gluing singleton traces into ℓ-traces. At large enough ℓ, GLUE can even outperform heavyweight defenses like Tamaraw. The overhead of GLUE varies, in the range of 3%–53%, depending on client behavior.

A web-browsing client is able to enlarge ℓ by simply increasing the maximum padding time. In fact, with a large enough maximum padding time, the client can consecutively visit webpages non-stop, and all the current best attacks will fail completely. Alternatively, we could implement a timer in the browser UI to remind the client not to dwell too long on a webpage: particularly privacy-sensitive clients could benefit from such a feature.

We propose that Tor's directory servers should maintain large databases of glue traces, and clients should load glue traces from them when necessary. We think it is a feasible scheme by showing that the extra distribution cost is quite low. It is worth investigating whether the client can generate glue traces "on the fly" that look like real web page traffic to eliminate this extra overhead and to ensure that the attacker cannot see the same traces as the client.

In this paper, we allow the attacker to know the entire database of glue traces. There are several reasons our attacker currently cannot pursue a strategy of simply identifying glue traces in the client's traffic. First, congestion and latency will perturb the glue trace, so that its instructions on when to send packets will not be exactly realized in the network trace, thwarting a simple matching attack. Second, glue traces are expected to be stopped prematurely by the client. Third, glue traces look like real web page loads, and the directory servers should maintain a large database of them; in other words, glue traces would look like real web page visits. As we cannot prove the impossibility of identifying glue traces in traffic, we leave the question open as future work; better counter-measures against it (such as limiting the attacker's knowledge of glue traces) are also possible.

Some other defenses can promise a certain level of guaranteed success against any WF attack, even future ones: among the practically deployable ones, Tamaraw has the lowest overhead, though it delays packets by 78% and almost doubles the bandwidth consumed. Considering the seemingly unavoidable overhead required, we did not design our defenses to guarantee future success. For example, we cannot prove that split decision and split finding are unsolvable, difficult as they are even with our improved CDSB. Many other practical defenses also cannot guarantee future success, including WTF-PAD and Tor Browser's randomized pipelining (which has recently been disabled). It remains to be seen whether future developments in the theory of traffic analysis can show what degree of guaranteed success FRONT and GLUE can achieve.

Acknowledgments

This work was partly funded by the Hong Kong Research Grants Council ECS Project Number 26203218.

Availability

We publish the simulation code used in this paper, including WF attacks we used and WF defenses we propose and evaluate in this paper. We also provide code used in split decision and finding. All the code and datasets are available via

https://github.com/websitefingerprinting/WebsiteFingerprinting/

References

[1] ABE, K., AND GOTO, S. Fingerprinting Attack on Tor Anonymity Using Deep Learning. *Proceedings of the Asia-Pacific Advanced Network* (2016).

[2] CAI, X., NITHYANAND, R., AND JOHNSON, R. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (2014), ACM.

[3] CAI, X., NITHYANAND, R., WANG, T., JOHNSON, R., AND GOLDBERG, I. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security* (2014), ACM.

[4] CAI, X., ZHANG, X. C., JOSHI, B., AND JOHNSON, R. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 19th ACM Conference on Computer and Communications Security* (2012), ACM.

[5] CHERUBIN, G. Bayes, not Naive: Security Bounds on Website Fingerprinting Defenses. *Proceedings on Privacy Enhancing Technologies* (2017).

[6] DYER, K. P., COULL, S. E., RISTENPART, T., AND SHRIMPTON, T. Peek-a-boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *33rd IEEE Symposium on Security and Privacy* (2012), IEEE.

[7] HAYES, J., AND DANEZIS, G. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *USENIX Security Symposium* (2016).

[8] HERRMANN, D., WENDOLSKY, R., AND FEDERRATH, H. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In *Proceedings of the 16th ACM Workshop on Cloud Computing Security* (2009), ACM.

[9] HINTZ, A. Fingerprinting Websites Using Traffic Analysis. In *International Workshop on Privacy Enhancing Technologies* (2002), Springer.

[10] JUAREZ, M., AFROZ, S., ACAR, G., DIAZ, C., AND GREENSTADT, R. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security* (2014), ACM.

[11] JUAREZ, M., IMANI, M., PERRY, M., DIAZ, C., AND WRIGHT, M. Toward an Efficient Website Fingerprinting Defense. In *European Symposium on Research in Computer Security* (2016), Springer.

[12] LI, S., GUO, H., AND HOPPER, N. Measuring Information Leakage in Website Fingerprinting Attacks and Defenses. In *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security* (2018), ACM.

[13] LIBERATORE, M., AND LEVINE, B. N. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (2006), ACM.

[14] LUO, X., ZHOU, P., CHAN, E. W., LEE, W., CHANG, R. K., AND PERDISCI, R. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *Network & Distributed System Security Symposium (NDSS)* (2011), Citeseer.

[15] OVERDORF, R., JUAREZ, M., ACAR, G., GREENSTADT, R., AND DIAZ, C. How Unique is Your. onion?: An Analysis of the Fingerprintability of Tor Onion Services. In *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM.

[16] PANCHENKO, A., LANZE, F., PENNEKAMP, J., ENGEL, T., ZINNEN, A., HENZE, M., AND WEHRLE, K. Website Fingerprinting at Internet Scale. In *Network & Distributed System Security Symposium (NDSS)* (2016), Citeseer.

[17] PANCHENKO, A., NIESSEN, L., ZINNEN, A., AND ENGEL, T. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society* (2011), ACM.

[18] RIMMER, V., PREUVENEERS, D., JUAREZ, M., VAN GOETHEM, T., AND JOOSEN, W. Automated Website Fingerprinting through Deep Learning. In *Network & Distributed System Security Symposium (NDSS)* (2018), Citeseer.

[19] SHMATIKOV, V., AND WANG, M.-H. Timing Analysis in Low-latency Mix Networks: Attacks and Defenses. In *European Symposium on Research in Computer Security* (2006), Springer.

[20] SIRINAM, P., IMANI, M., JUAREZ, M., AND WRIGHT, M. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security* (2018), ACM.

[21] SYVERSON, P., DINGLEDINE, R., AND MATHEWSON, N. Tor: The Second Generation Onion Router. In *USENIX Security Symposium* (2004).

[22] TOBIAS PULLS. Adaptive Padding Early (APE). `https://www.cs.kau.se/pulls/hot/thebasketcase-ape/`, 2016. [Online; accessed 25-August-2018].

[23] WANG, T. Optimizing Precision for Open-World Website Fingerprinting. *arXiv preprint arXiv:1802.05409* (2018).

[24] WANG, T., CAI, X., NITHYANAND, R., JOHNSON, R., AND GOLDBERG, I. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security Symposium* (2014).

[25] WANG, T., AND GOLDBERG, I. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society* (2013), ACM.

[26] WANG, T., AND GOLDBERG, I. On Realistically Attacking Tor with Website Fingerprinting. *Proceedings on Privacy Enhancing Technologies* (2016).

[27] WANG, T., AND GOLDBERG, I. Walkie-Talkie: An Efficient Defense against Passive Website Fingerprinting Attacks. In *USENIX Security Symposium* (2017).

[28] WRIGHT, C. V., COULL, S. E., AND MONROSE, F. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *Network & Distributed System Security Symposium (NDSS)* (2009), Citeseer.

[29] XU, Y., WANG, T., LI, Q., GONG, Q., CHEN, Y., AND JIANG, Y. A Multi-tab Website Fingerprinting Attack. In *Proceedings of the 34th Annual Computer Security Applications Conference* (2018), ACM.

[30] ZHUO, Z., ZHANG, Y., ZHANG, Z.-L., ZHANG, X., AND ZHANG, J. Website Fingerprinting Attack on Anonymity Networks Based on Profile Hidden Markov Model. *IEEE Transactions on Information Forensics and Security* (2018).

A How to Set FRONT Parameters

There are four main parameters in FRONT, namely, N_c, N_s, W_{min} and W_{max}. Obviously, $N_c + N_s$ determines the data overhead. It is worth considering how to set the ratio between them two, given a fixed data overhead. We also investigate how to set W_{min} and W_{max} in the following.

A.1 Impact of Padding Budget Ratio

We want to investigate the optimal ratio between N_c and N_s given a fixed total data overhead. We define a padding budget ratio $\alpha = N_c/(N_c + N_s)$, which is the proportion of total padding used by the client. We set $W_{min} = 1\,\text{s}, W_{max} = 14\,\text{s}$, and $N_s + N_c = 5000$ based on FT-2. This results in a data overhead of 49%.

Figure 14 shows attack performances with different α. In the figure, each line represents an attack. Each attack's performance has an inflection point as we increase α; we mark the optimal α using a black dot, i.e. the value at which each attack is least effective.

The upper figure shows how TPR changes for the three attacks based on α. The TPR achieved by each attack greatly decreases under FRONT, especially for kNN and CUMUL. We found that the optimal α values are 0.32, 0.32, 0.5 and 0.24 in terms of TPR for the three attacks. The lower figure shows the change of precision. Precision curves exhibit greater fluctuation. The optimal α values are still around 0.5 except for kFP. Combining these results, we find that the optimal α is around 0.25–0.5. This suggests that we should set N_s to be equal to or a bit smaller than N_c.

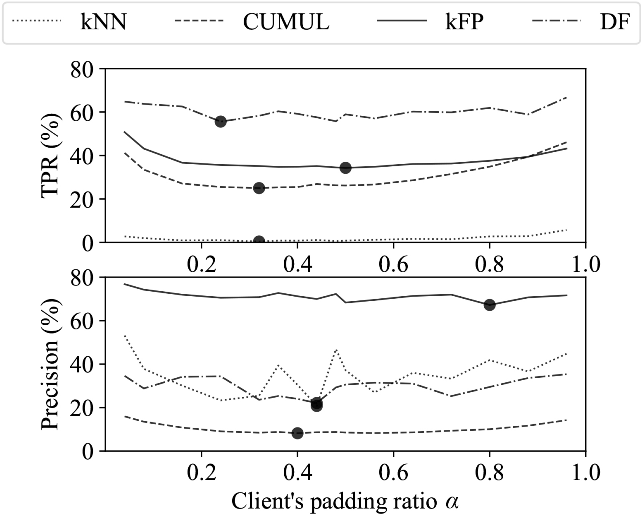

Figure 14: Three WF attacks' performances with different α. The upper figure shows attack results in terms of their TPR. The lower figure shows precision. We point out the optimal ratio for our defense using a black dot in each subfigure.

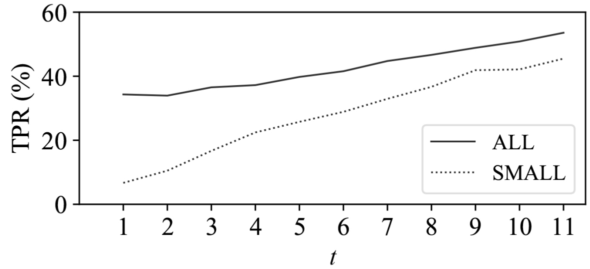

Figure 15: TPR of kFP with different t. The full line shows TPR on the full dataset while the dotted line shows TPR on small webpages with mean loading time less than 20 s.

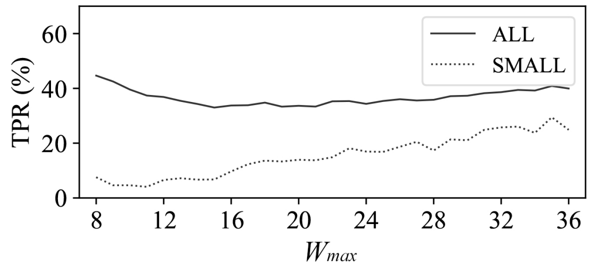

Figure 16: TPR of kFP with different W_{max}. The full line shows TPR on the full dataset while the dotted line shows TPR on small webpages with loading time less than 20 s.

A.2 Set the Padding Window Parameters

In our design, the padding windows for both client and server are sampled from $U(W_{min}, W_{max})$. We introduce W_{min} to ensure that the real padding window is not too small to satisfy the network bandwidth. So how do we set W_{max}? Intuitively, with a larger W_{max}, the range of possible padding window size is larger, resulting in more randomness. However, this may also cause a "long tail" of Rayleigh distribution — more dummy packets are scheduled to the end of the trace or even dropped due to FRONT design. This may reduce the security level of FRONT, especially for small webpages. We did two experiments to validate this.

Enlarge W_{min} and W_{max} In this experiment, we try to investigate the impact of enlarging padding window size. We set $N_s = N_c = 2500$ based on FT-2. Varying t from 1 s to 11 s, we set $W_{min} = t$ and $W_{max} = 13 + t$. This means that we are enlarging the expected window size under the same randomness since the maximum change of sampled padding window size is always within $W_{max} - W_{min} = 13\,\text{s}$. We use kFP as the attacker since it consistently achieves the best performance against FRONT, as is shown in Section 5. We show TPR on the full dataset as well as on small webpages whose mean loading time is less than 20 s. The small webpages account for 16% in our dataset and the mean loading time on the whole dataset is 27 s. Figure 15 shows the result. Both lines keep increasing when the expected window size is enlarged. This again validates our intuition that it is better to

Table 5: Feature set of split decision.

No.	Feature description
1	Transmission size
2	Transmission time
3	Number of outgoing packets
4–5	Mean, std of inter-arrival times
6–105	Top 100 inter-arrival times
106–107	Mean, std of top 100 inter-arrival times
108–111	25, 50, 75 and 100 percentile of top 100 inter-arrival times

have more packet padded in the trace front as well as avoid packet dropping in the trace end.

Enlarge W_{max} only In the second experiment, we try to find out how to set W_{max} after we decide W_{min}. We fix $W_{min} = 1$ s but vary W_{max} from 14 s to 36 s. Figure 16 shows the result. On the full dataset, TPR decreases from 45% to 33% at 15 s and then bounce back to 40%. However, TPR on small webpages keeps increasing from 8% to 30%. This indicates that for small webpages, most dummy packets are left unused in the end since their timestamps are too large, resulting in the increase in TPR. As for other webpages, the randomness accounts for the decrease first while the drop of dummy packets dominates the randomness after $W_{max} > 15$ s, leading to the increase in TPR.

To conclude, we should set W_{max} reasonably large to achieve good randomness. But we can not make W_{max} too large to avoid dropped dummy packets. For simplicity of our design, we set a global W_{max} for all webpages. (Therefore, we set $W_{max} = 14$ s in our experiments.) But if we are allowed to have some information about webpages, it will be better to have a dynamic W_{max}.

B Split Decision Features

Features used in split decision. Feature 1 and 3 are volume information while the others are time information of a trace. The first 3 features help us determine how many webpages in an ℓ-trace by the length of the trace. Feature 4–111 extract information from large gaps in an ℓ-trace. They help determine how many splits are in the trace.

C Score Decoding Algorithm

Algorithm 1 shows the pseudocode of score decoding process. The inputs are scores for all outgoing packets, the number of splits to be found and a parameter neighborhood r. We find one split in each round by picking out the highest score while masking all outgoing packets in the "neighborhood". In other words, neighbor packets will not be considered in the following rounds. We set $r = 40$ in our experiments.

Algorithm 1 Score Decoding

```
Input:
    A list containing each outgoing packet's location and score;
    A parameter: Neighborhood $r$;
    The number of splits $n$;
Output:
    Set of predicted splits $L$;
$L \leftarrow \{\}$;
for $i = 1$ to $n$ do
    Find the packet $p$ with highest score and add it into $L$;
    Set $p.score \leftarrow -\infty$;
    for every other packet $q$ do
        if $|q.loc - p.loc| < r$ then
            $q.score \leftarrow -\infty$;
        end if
    end for
end for
return $L$;
```

D Information Leakage Analysis Result

In Section 5.3, we show the ECDF of information leakage. Here we present the detailed result of information leakage analysis. We estimate information leakage for 3043 features on both undefended and defended traces. These features are grouped into 14 different categories and they have covered all the features WF attacks use in the literature [12]. Figure 17 shows the information leakage for each feature on our datasets in the open-world scenario.

FRONT results in less information leakage in most of the categories compared to WTF-PAD, especially for features like *Pkt. Count*, *Time*, *NGRAM*, *Pkt. Distribution* and *CUMUL*. WTF-PAD outperforms FRONT in category *Interval-I*, *II* and *III*. This result makes sense since WTF-PAD is based on obfuscating time features while FRONT focuses mainly on obfuscating volume features as well as bringing in more randomness.

E Evaluation on Undefended ℓ-traces

Figure 18 and 19 shows the attack TPR and precision on undefended ℓ-traces, without and with split decision, respectively. When ℓ is known (i.e., without split decision), all attacks achieve similar TPR under all the ℓ values. But precision varies. kFP has the highest precision all four attacks all the time, ranging from 82% to 97%. When ℓ is unknown (i.e., with split decision), TPR and precision of all attacks except kNN drop by 20–30%, but still share the same trend as when without split decision. kNN's performance is greatly affected by split decision when $\ell \geq 9$.

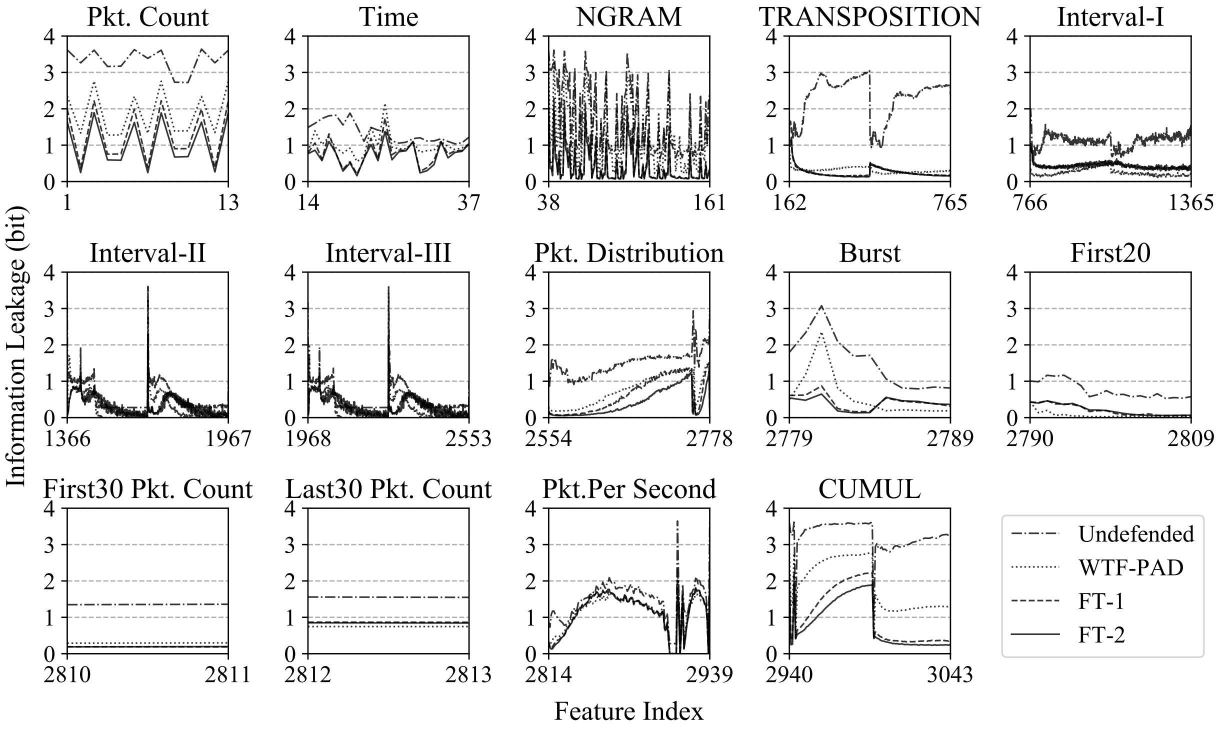

Figure 17: Information leakage for individual features.

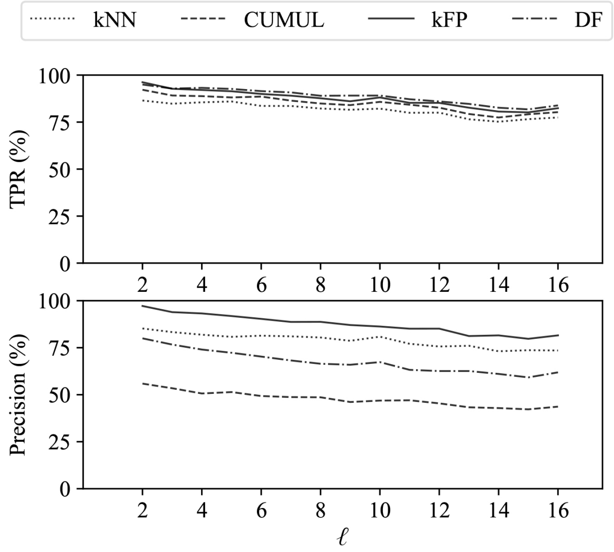

Figure 18: WF attack performance without split decision on clean ℓ-traces.

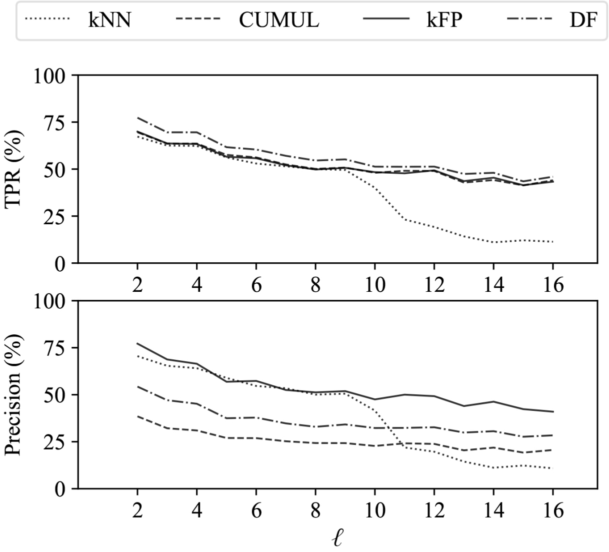

Figure 19: WF attack performance with split decision on clean ℓ-traces.