James K Holland* and Nicholas Hopper

# RegulaTor: A Straightforward Website Fingerprinting Defense

**Abstract:** Website Fingerprinting (WF) attacks are used by local passive attackers to determine the destination of encrypted internet traffic by comparing the sequences of packets sent to and received by the user to a previously recorded data set. As a result, WF attacks are of particular concern to privacy-enhancing technologies such as Tor. In response, a variety of WF defenses have been developed, though they tend to incur high bandwidth and latency overhead or require additional infrastructure, thus making them difficult to implement in practice. Some lighter-weight defenses have been presented as well; still, they attain only moderate effectiveness against recently published WF attacks. In this paper, we aim to present a realistic and novel defense, RegulaTor, which takes advantage of common patterns in web browsing traffic to reduce both defense overhead and the accuracy of current WF attacks. In the closed-world setting, RegulaTor reduces the accuracy of the state-of-the-art attack, Tik-Tok, against comparable defenses from 66% to 25.4%. To achieve this performance, it requires 6.6% latency overhead and a bandwidth overhead 39.3% less than the leading moderate-overhead defense. In the open-world setting, RegulaTor limits a precision-tuned Tik-Tok attack to an $F_1$-score of .135, compared to .625 for the best comparable defense.

**Keywords:** website fingerprinting, traffic analysis

DOI 10.2478/popets-2022-0049
Received 2021-08-31; revised 2021-12-15; accepted 2021-12-16.

## 1 Introduction

The low-latency anonymity network Tor protects the privacy of its users' internet browsing habits, allowing them to evade surveillance, tracking, and censorship. To do this, it encrypts internet traffic while routing it through a series of volunteer-run nodes, preventing any single node from knowing both the origin and destination of the communications [19]. As a result, traffic destinations cannot determine the identities of their users, and local observers (such as network administrators or ISPs) cannot see the destinations of users' traffic.

In the last decade, Tor's user base has increased to millions of daily users [9]; over the same period, a series of papers have shown that Tor is vulnerable to a type of traffic analysis attack known as *website fingerprinting* (WF) [17, 22, 27, 35, 38, 42, 45]. In a WF attack, a passive eavesdropper attempts to determine the destination of encrypted traffic by observing the sequences of packets sent and received by the user. The attack takes place between the user and the first node in the Tor network using features such as the volume of incoming and outgoing packets and the relative timing of packet bursts. With this information, the attacker then compares the collected packet trace to a database of website-trace pairs and classifies each trace accordingly.

WF attacks are straightforward to carry out, as they require only passive eavesdropping from a local adversary, ISP, or Tor guard. Thus, WF attacks against Tor users represent a realistic threat and may be used to identify users who visit 'censored' or forbidden websites. As this poses an obvious threat to users' privacy, researchers have developed a variety of defenses [16, 17, 21, 29, 30, 46, 47]. The goal of these defenses is to alter traffic in a manner that makes it difficult to determine which website is associated with each packet trace, and they typically operate by strategically adding 'dummy' packets or by delaying packets.

However, the Tor Project has been hesitant to implement past defenses, as most would either impact user experience with increased latency, burden the Tor network with increased bandwidth, or require the creation and maintenance of additional infrastructure. Additionally, many of these defenses have been proven ineffective against the latest attacks, which utilize large data sets and sophisticated deep learning techniques. In response, we present RegulaTor, which provides strong protection against state-of-the-art WF attacks with moderate bandwidth overhead and a small latency penalty, but without requiring additional infrastructure or knowledge of other traces.

***Corresponding Author: James K Holland:** University of Minnesota, E-mail: holla556@umn.edu
**Nicholas Hopper:** University of Minnesota, E-mail: hoppernj@umn.edu

Our key observation is that defenses that "regularize" traffic so that traces from different web pages are identical tend to be the most effective. However, these defenses are often the least efficient, as they incur high latency overhead in periods of heavy traffic and high bandwidth overhead in periods of light traffic. Still, constant rate traffic is *just one of many* potential patterns for traffic regularization. Based on empirical evaluation of Tor web traffic, we find that there are common traffic patterns that avoid these traffic rate mismatches, allowing users to achieve the security benefits of regularization while greatly reducing the associated overhead.

Accordingly, RegulaTor works by regularizing the size and shape of packet 'surges' that frequently occur in download traffic, masking potentially revealing features. In this paper, 'surge' is broadly defined as a large number of packets sent over a short period of time. To do this, whenever a download traffic 'surge' arrives, RegulaTor starts sending packets at a set initial rate to avoid leaking information about the volume and length of the surge. Then, it decreases the packet sending based on a set 'decay rate' parameter, which defines the shape of the surge. If no packets are available when one is scheduled, a dummy packet is sent instead. However, due to the heavy 'burstiness' of web-browsing traffic, this download padding approach can be carried out with limited overhead. At the same time, RegulaTor sends upload packets at some fraction of the download packet sending rate. Moreover, sending upload packets based on download traffic usually incurs little latency overhead, as upload traffic mimics the download traffic (albeit with less volume) in web browsing traffic, as shown later in this paper.

The RegulaTor approach deviates significantly from previously proposed WF defenses. First, it alters traffic in a time-sensitive manner with a focus on sending standardized surges, while other defenses (with FRONT [21] a notable exception) tend to insert padding consistently along the packet sequence. Furthermore, it uses entirely different strategies to alter upload and download traffic, while other defenses pad traffic consistently regardless of direction. Lastly, RegulaTor uses the observed similarity between upload and download traffic to send upload traffic as a function of download traffic, preventing upload traffic from leaking any further information.

We also re-evaluate previously presented attacks and defenses on our data set to enable direct comparison. In the closed-world setting with 95 websites, RegulaTor reduces the accuracy of the state-of-the-art attack, Tik-Tok, to only 25.4% compared to 66.0% for FRONT-2500. Furthermore, it requires a latency overhead of 6.6% and a bandwidth overhead of only 79.7%, while FRONT-2500 requires a bandwidth overhead of 119%. In the open-world setting, RegulaTor's performance advantage is even more severe, reducing the $F_1$-score of a precision-tuned Tik-Tok attack to .135 compared to .625 for FRONT-2500. Thus, RegulaTor drastically outperforms comparable defenses in terms of efficiently defending traffic in the open-world setting.

# 2 Preliminaries

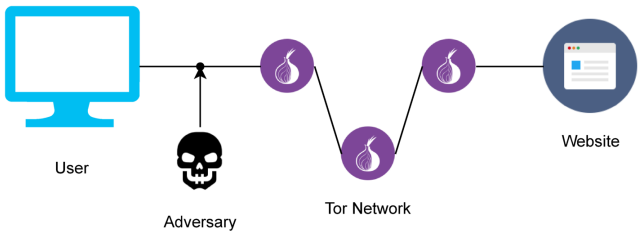
## 2.1 Website Fingerprinting Background

For WF attacks, the threat model is a passive, local adversary who eavesdrops on network traffic. Potential attackers include ISPs, network administrators, and the guard node of the Tor network, as shown in Figure 1. However, the attacker is not able to modify traffic in any way. To carry out the attack, the attacker first collects the traffic trace associated with the target's web browsing. Then, the attacker compares the trace to a data set of (trace, web page) pairs, finding the matching web page based on a chosen set of features.

WF attacks are typically tested in two different settings: *closed-world* and *open-world*. In the closed-world setting, we assume that the target visits one of a defined set of web pages, and the attacker simply has to determine which of those web pages was visited. Because a real-world target may visit any number of web pages, this setting is unrealistic. Still, the closed-world accuracy is useful for comparing and evaluating defenses against a relatively strong WF attacker.

On the other hand, the open-world setting provides a much more realistic setting where the target can visit any number of web pages. In this case, the attacker's goal is to determine whether or not the target visited a *monitored* web page using information from the packet

**Fig. 1.** WF Attack on Tor



trace. While this task may be significantly more difficult given that the attacker cannot possibly train a model on all possible web pages, it is much more similar to the real-world scenario of an adversary trying to catch users visiting censored web pages.

Furthermore, we make two major assumptions about the attacker's abilities as originally stated by Juarez et al. [28]. First, we assume that the attacker can detect the beginning and end of the page load, which is necessary to store a packet sequence representative of a given web page. Then, we also assume that the attacker can create a data set representing the targets's unique conditions. Some of these conditions include the Tor Browser Bundle (TBB) version, operating system, device hardware, and geographic location. Because these assumptions all favor the attacker, we expect that real-world defense performance is at least as high as demonstrated in this paper.

Hintz et al. [25] were the first to demonstrate success in WF attacks using the set of file transfer sizes to distinguish web pages. While this attack appeared to be effective against other privacy and anonymity systems, such as VPNs [24, 40], Tor at first appeared to be immune due to constant-size cell-padding, circuit multiplexing, and network-induced delays. However, later attacks utilized packet volume and timing features to greatly increase WF attack effectiveness against Tor [17, 35].

Wang et al. then made further improvements [45], first by improving data gathering and pre-processing, and later by using a large feature set with the k-nearest neighbors algorithm [44]. Later, the k-fingerprinting approach [22], which used a series of simple but important features along with random forests and k-nearest neighbors, further improved accuracy. But, most importantly, k-fingerprinting demonstrated effectiveness in an open-world setting with a world size much larger than in previous works. Soon afterward, The CUMUL approach [34] was presented by Panchenko et al. By sampling features from the cumulative representation of a trace, CUMUL outperforms previous attacks while staying computationally efficient.

While these models demonstrate high accuracy, the Deep Fingerprinting approach by Sirinam et al. [42] uses convolutional neural networks to further improve the state-of-the-art in WF attack performance. Most importantly, Deep Fingerprinting manages to defeat the WTF-PAD defense and achieve high accuracy against the Walkie-Talkie defense in the closed-world scenario. As a result, this paper uses Deep Fingerprinting as a benchmark to test the RegulaTor defense and its rivals. The Tik-Tok attack [38] further improves the Deep Fingerprinting attack by using burst-level timing features. While previous attacks have ignored granular timing information, Tik-Tok uses the combination of directional and timing features to further improve performance against WF defenses, including the Walkie-Talkie defense. Accordingly, we use Tik-Tok as a benchmark in this paper as well. Other WF attacks and related techniques are discussed in the related works section.

Defenses evaluated in this paper include Tamaraw, WTF-PAD, and FRONT. Tamaraw [16] serves as a baseline defense that regularizes traffic while providing proven security guarantees. However, it is not a practical defense, as it requires high bandwidth and latency overhead. WTF-PAD [28] uses adaptive padding to fill gaps in packet sequences to reduce the amount of information leaked by each trace. Alternatively, FRONT [21] adds varying amounts of dummy packets to the beginning of packet sequences, making them more difficult to distinguish. Both WTF-PAD and FRONT require moderate bandwidth overheads and add no latency. Moreover, neither WTF-PAD nor FRONT require additional infrastructure or detailed descriptions of other traces; thus, they are suitable comparisons for RegulaTor, which can also be implemented in a relatively straightforward manner. For a more in-depth summary of WF defenses, see the related works section.

## 2.2 Metrics

In the closed-world setting, the evaluation is straightforward, as the attacker simply needs to determine the correct website with the highest possible accuracy. However, the open-world setting is more complicated due to the imbalanced classes, as the attacker may only be able to monitor a small portion of web pages a user could visit. Thus, there is a strong potential for false positives to outnumber the true positive results. This possibility was discussed in the Tor community [2], and attacks have since aimed to maximize the *precision* of their classifiers. In the open-world scenario, the precision represents the monitored web pages that were detected compared to all of the web pages predicted to be in the monitored set. Accordingly, precision is a much more useful metric for evaluating the usefulness of a WF attack in a realistic scenario. Alternatively, recall, representing the fraction of monitored web pages that were retrieved, is also used in the open-world setting to demonstrate the sensitivity of an attack. An ideal attack has both high precision and high recall; however, one often comes at the expense of the other.

## 2.3 Defense Overhead

WF defenses have the potential to impact user experience and increase strain on the Tor network by increasing latency, delaying page loading, and increasing the required bandwidth. As a result, we evaluate the *latency overhead* and *bandwidth overhead* of each defense. We find latency overhead by calculating the additional time required to send the *defended* trace compared to the undefended trace and dividing by the time required to send the original trace. In practice, we do this by subtracting the sending time of the last *real* packet in the defended trace from the sending time of the last packet in the undefended trace. Intuitively, the latency overhead metric aims to indicate how much longer the user will have to wait to load a web page.

The bandwidth overhead is found by dividing the number of dummy packets sent in the defended trace by the number of packets in the undefended trace. Essentially, it represents how much more data will be transmitted while loading a web page with the WF defense enabled.

Though increased bandwidth may strain the Tor network, discussion in the Tor community indicates that it is preferable to latency overhead. This is because a reasonable degree of bandwidth overhead can be managed by Tor without affecting the user experience, but increased latency would increase page load time and potentially contradict Tor's position as a 'low latency' protocol [1].

As a result, the RegulaTor defense design aims to defeat WF attacks primarily through increased bandwidth, while delaying traffic only when the increased packet sending rate would 'leak' a significant amount of information about the target web page. Due to its importance, latency overhead is one of the primary metrics used to evaluate RegulaTor and comparable defenses.

## 2.4 Data Sets

Our primary evaluation uses two data sets provided by Sirinam et al. that were originally collected in 2016 to test their Deep Fingerprinting attack [41, 42]. The closed-world data set was collected by visiting the homepages of the Alexa Top 100 sites 1,250 times each using tor-browser-crawler on ten low-end machines [8]. The homepage visits were split into five batches where for each batch, each machine would access a website 25 times before moving on to the next website. Batching the crawling in this manner controls for both long and short-term variance, as described by Wang et al. [45]. After discarding corrupted traces, 1000 instances of 95 sites were included in the final data set, which we refer to as DF-CW.

The open-world data set was collected using tor-browser-crawler [8] to visit the sites in the Alexa Top 50,000, excluding the top 100 sites crawled for the closed-world data set. Again, ten low-end machines were used, with each machine making one visit to the home pages of 5000 different sites. After discarding corrupted visits, 40,716 traces were included in the open-world data set, which we refer to as DF-OW.

Other data sets include the recently collected Goodenough data set provided by Pulls [36] and the cell traces provided by Wang et al. [44] to demonstrate the k-nearest neighbors attack. Both are used to test RegulaTor's generalizability. The former data set contains data for 100 websites with 90 instances each and was collected in 2014. The latter data set, created in 2020, collected 20 samples from each of 10 web pages for 50 websites, resulting in a total of 10,000 samples. For both data sets, only the closed-world traces are used for straightforward evaluation and comparison between defense settings. In this paper, we refer to the k-nearest neighbors data set as KNN and the Goodenough data set as GE.

Furthermore, to test RegulaTor parameters in a real-world implementation, we use our pluggable transport implementation to collect 100 defended samples from each of the websites in the Alexa top 100. This data set was collected over one month in August 2021 and is referred to as PT.

# 3 RegulaTor

## 3.1 RegulaTor Justification

Given that the majority of relatively efficient previous WF defenses have been defeated, we aim to design a defense that can resist WF in both open-world and closed-world settings. We aim to keep the implementation simple and functional without requiring frequent tuning or collection of data sets, which is generally required in machine learning-based defenses. While this task is difficult, we find that there are common traffic patterns in Tor traffic that allow for an effective defense to operate with moderate overhead.

First, we analyzed DF-CW and found that the **packet traces are particularly surge-heavy** in that they consist of infrequent and irregular 'surges' of pack-

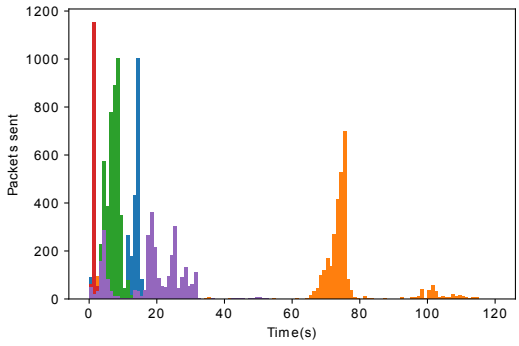



**Fig. 2.** Examples of Tor download traffic during web page visits

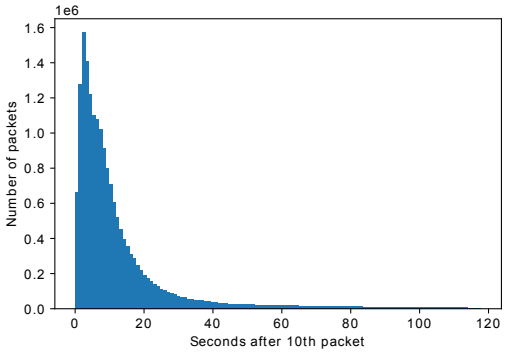



**Fig. 3.** Decay of undefended download packet sending volume

ets sent with little inter-packet delay. To illustrate this, we have plotted the download packet timing patterns from several randomly chosen traces (colored to distinguish between them) in Figure 2. Note that the traffic is characterized by occasional high-volume packet surges followed by periods of low traffic. We also find that, even though the average web page visit in DF-CW lasts about 28 seconds, the median interquartile range of packet times is only 3.96 seconds. This further demonstrates that the bulk of traffic is sent over a relatively short period of time. Furthermore, the location, size, and timing of these surges represent coarse features that leak a significant amount of information about the traces. To prevent these coarse features from leaking information about the traces, padding can be done in a more randomized manner to obfuscate the features, which was done in the FRONT defense; or, the surges in the packet sequences can be regularized in terms of size and location, which is the approach presented in this paper.

Luckily, the packet sequences show that **surge patterns are often predictable**. To be specific, a majority of the packet sequences consist of an early sequence of upload packets followed by a sudden surge of download packets. While the download surge varies in terms of size and start time, it generally decays in volume soon after the initial spike as the web page finishes loading. This is shown in Figure 3, which represents the packet distribution for 10,000 randomly chosen traces. In order to control for the start time of the first surge, only traffic after the 10th packet is shown. Additionally, the median packet is sent 7.57 seconds after the 10th packet, further emphasizing how the bulk of traffic is sent soon after the beginning of a web page visit. Accordingly, regularizing the download packet sequences can be done efficiently by adding dummy packets to sequences with smaller initial surges and delaying packets in sequences with larger initial surges.

Furthermore, the **timing of upload packets imitates the timing of download packets**, despite the relatively low volume of upload traffic, as outgoing requests are generally quickly responded to by the web page. The correlation between upload and download traffic volume is illustrated in Figure 4, where the traffic from 30 randomly chosen traces is split into 1-second bins and plotted based on the number of upload and download packets sent in that period. Most importantly, Figure 4 shows that if a substantial number of download packets are being sent, then upload packets are being sent concurrently. This presents an opportunity for RegulaTor to defend the upload packet sequence as well: by modeling upload packet sending as a function of the download packet sequence, the upload traffic leaks no further information about the destination web page. Additionally, RegulaTor can minimize upload packet latency increases by intentionally overestimating the upload sending rate, which is relatively efficient given the lower volume of upload packets.

For a visual representation of the RegulaTor defense, see Figure 5, which illustrates the rate of download and upload packet sending at different points in time for a single defended web page visit.

## 3.2 RegulaTor Design

To enact the defense, the client pads the upload packets, while the download padding can be carried out by a Tor bridge, middle node, or guard node. However, padding to the middle node is likely the most effective method, as a WF adversary may be located at the guard

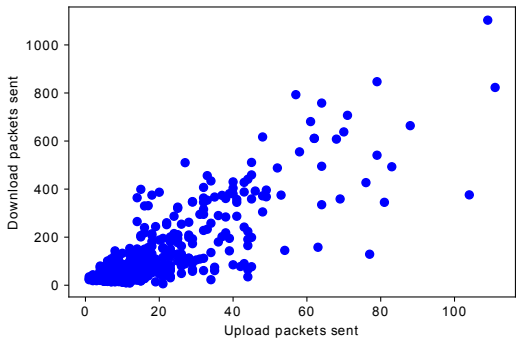



**Fig. 4.** Download vs. upload traffic for each second

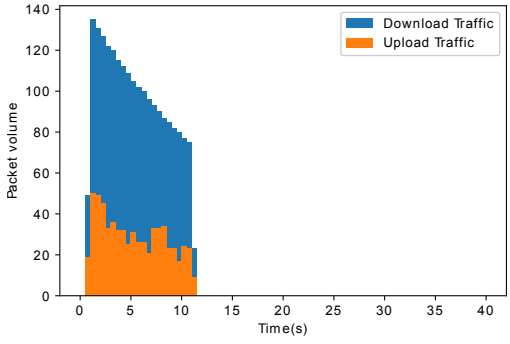



**Fig. 5.** RegulaTor-defended Trace

| Parameters | Description |
|---|---|
| R | Initial surge rate (packets $s^{-1}$) |
| D | Packet sending decay rate ($s^{-1}$) |
| T | Surge threshold ratio |
| N | Padding budget (packets) |
| U | Download-upload packet ratio |
| C | Delay cap (s) |

**Table 1.** RegulaTor parameter descriptions

node. By only padding the traffic between the client and one of the first relays in the circuit, the real-world cost of the bandwidth overhead can be substantially minimized. In fact, Tor is generally bandwidth-limited by the relatively limited number of exit nodes [29], further minimizing the impact that RegulaTor-defended traffic will have on the network.

The RegulaTor defense pads the download packets as follows: the first download packets are sent at a constant rate until 10 packets have been sent. This is done to avoid sending the initial RegulaTor surge before the surge of original data has been scheduled while also allowing the circuit-building and TLS handshake to finish. At this point, RegulaTor begins to send a surge of packets at the initial surge rate, $R$, which represents the number of packets sent per second. However, the sending rate is reduced according to the decay constant, $D$, such that the sending rate is $RD^t$, where $t$ is the number of seconds since the surge began. Afterward, if the original packet sequence again calls for a significant number of packets to be sent and the queue of waiting packets increases to some threshold, then RegulaTor sends another surge of download packets to minimize the potential for increased latency. This threshold is calculated as the surge threshold, $T$, multiplied by the target rate.

To minimize additional bandwidth overhead, the defense draws a random padding budget from $(0, N)$, where $N$ is the maximum padding size. Once $N$ dummy packets have been sent, the sending of dummy packets is stopped, though real packets may still be delayed. The random choice of $N$ also functions to vary the total volume of the packet sequences, which reduces the amount of information leaked by the volume of the sequence.

RegulaTor pads the upload trace at a constant rate until the initial download surge begins to arrive to accommodate the initial web page request. At this point, the client schedules upload packets to send at some ratio, $U$, of the download packet sending rate (e.g. the client will send upload packets at 1/3 the rate of the download packets for $U = 3$). $U$ is chosen to minimize upload packet sending latency, so it will typically send a significant number of dummy packets; however, this is not costly bandwidth-wise, as the upload packet sequences are typically small. Additionally, if any upload packets have been delayed for more than $C$ seconds, then they are sent immediately to prevent excessive latency.

Algorithm 1 demonstrates how the download packing sending schedule is determined, and table 1 contains the relevant parameters and their descriptions.

## 3.3 Parameter Tuning

To determine the specific parameter values for RegulaTor, we used the Tree-Structured Parzen Estimator (TPE) technique, which is usually used for hyperparameter optimization for learning algorithms [13]. To implement parameter tuning, we used the Python library hyperopt [5] and determined RegulaTor's performance by simulating RegulaTor on DF-CW and testing the performance of Tik-Tok [38] against the RegulaTor-

**Algorithm 1** RegulaTor download padding main loop

```
while < 10 packets scheduled do
  wait
end while

surge-time ← CURRENT-TIME
next-packet-time ← CURRENT-TIME
while web page downloading do
  if CURRENT-TIME ≥ next-packet-time then

    target-rate ← RD^(CURRENT-TIME − surge-time)

    if target-rate < 1 then
      target-rate ← 1
    end if

    if waiting-packets > T · target-rate then
      surge-time ← current-time
    end if

    if NUM-WAITING-PACKETS = 0 then
      if sent-dummy-packets < N then
        SEND-DUMMY-PACKET
        sent-dummy-packets ← sent-dummy-packets + 1
      end if
    else
      SEND-PACKET
    end if

    time-gap ← target-rate^−1
    next-packet-time ← next-packet-time + time-gap
  end if
end while
```

defended data set. We chose Tik-Tok as the WF attack so that RegulaTor would be tuned to avoid leaking timing information, which Tik-Tok can detect with considerable effectiveness.

Then, to determine the performance of parameter combinations, we created a loss function based on a weighted combination of the latency overhead, bandwidth overhead, and WF attack accuracy. This also allows us to alter RegulaTor based on which properties (e.g. low latency) are most desired. In our evaluation, we present two RegulaTor defenses: one that achieves high performance at the expense of increased overhead (RegulaTor-Heavy), and another that aims to achieve moderate performance with lower overhead (RegulaTor-Light). Further details regarding the parameter tuning are available in the appendix.

# 4 Defense Evaluation

In this section, we evaluate RegulaTor and several other WF defenses in terms of their closed-world performance, open-world performance, and overhead. Table 2 presents the parameters used in each of the examined defenses. For defenses other than RegulaTor, default parameters were used as provided by their authors [16, 21, 29].

For evaluation, we simulated each defense on the undefended data sets DF-CW and DF-OW to create defended data sets for the closed-world and open-world settings. To simulate WTF-PAD and FRONT, we used the code provided by their authors. For Tamaraw, we used Tao Wang's implementation [3]. Then, we retrained each WF attack on the defended data sets. The attacks used in this section used default parameters as well, except for CUMUL, which performs SVM hyperparameter tuning.

To demonstrate RegulaTor's generalizability, we also simulate RegulaTor on data sets used in past works, present the performance of a real-world RegulaTor implementation, and investigate parameter tradeoffs. Then, we discuss parameter stability over time and the practicality of RegulaTor deployment.

## 4.1 Closed-World

Table 3 presents the accuracy achieved by WF attacks on the examined defenses. Since the experiment was done in the closed-world setting, accuracy is the only metric presented, and the DF-CW data set was used. Each attack achieved high accuracy on the undefended data set with Deep Fingerprinting achieving the highest accuracy. Tik-Tok and Deep Fingerprinting were highly effective against WTF-PAD, demonstrated moderate effectiveness against FRONT defenses, and were only marginally effective against RegulaTor defenses. However, CUMUL accuracy sharply decreased for all defenses. No attack was effective against Tamaraw, which is included as an example of a defense with strong theoretical foundations but impractically high overhead.

Tik-Tok outperformed Deep Fingerprinting against the RegulaTor defense, likely due to the use of packet timing to provide further information. Given that RegulaTor generally sends upload packets at regular intervals, attacks that represent the traces using only packet direction, such as Deep Fingerprinting, are unable to achieve high accuracy. The usefulness of timing information was most apparent with RegulaTor-Light, which

| Defenses | Parameters |
|---|---|
| Tamaraw | $\rho_{out} = .04, \rho_{in} = .012, L = 100$ |
| WTF-PAD | normal_rcv |
| FRONT-1700 | $N_s = N_c = 1700, W_{min} = 1, W_{max} = 14$ |
| FRONT-2500 | $N_s = N_c = 2500, W_{min} = 1, W_{max} = 14$ |
| RegulaTor-Light | $R = 260, D = .860, T = 3.75, N = 2080, U = 4.02, C = 2.08$ |
| RegulaTor-Heavy | $R = 277, D = .940, T = 3.55, N = 3550, U = 3.95, C = 1.77$ |

**Table 2.** Defense Parameters

| Defenses | Tik-Tok | DF | CUMUL |
|---|---|---|---|
| Undefended | 97.0% | 98.4% | 97.2% |
| WTF-PAD | 94.2% | 92.4% | 59.4% |
| FRONT-1700 | 78.2% | 77.5% | 31.6% |
| FRONT-2500 | 66.0% | 69.8% | 17.1% |
| RegulaTor-Light | 34.8% | 23.3% | 20.8% |
| RegulaTor-Heavy | 25.4% | 19.6% | 16.3% |
| Tamaraw | 10.1% | 9.9% | 17.0% |

**Table 3.** Closed-World Accuracy

reduced Tik-Tok accuracy to 34.8% compared to 23.3% for Deep Fingerprinting. However, it should be noted that the RegulaTor parameters were tuned based on defense performance against Tik-Tok, while WTF-PAD and the FRONT defenses used default parameters. As a result, the RegulaTor defenses may have had a slight advantage against Tik-Tok.

Both RegulaTor-Heavy and RegulaTor-Light reduced Tik-Tok and Deep Fingerprinting accuracy significantly more than any other practical defense. Even when comparing the 'light' version of RegulaTor to the bandwidth-heavy version of FRONT, Tik-Tok accuracy is decreased from 66.0% to 34.8% and Deep Fingerprinting accuracy is decreased from 69.8% to 23.3%. However, CUMUL accuracy is not necessarily decreased when comparing FRONT to RegulaTor-Light. We suspect that this is because CUMUL derives a majority of its features from the cumulative representation of the trace, and FRONT effectively obfuscates early incoming and outgoing bursts in the trace, modifying the cumulative representation substantially.

Furthermore, RegulaTor-Light manages to outperform its rivals while incurring a small latency overhead and a bandwidth overhead similar to that of WTF-PAD. RegulaTor-Heavy then further increases this margin with a bandwidth overhead still less than that of the lower-overhead version of FRONT.

## 4.2 Open-world

### 4.2.1 Open-World Setup

While the closed-world setting is useful for illustrating the relative effectiveness of WF attacks and defenses, it is not a particularly strong indicator of real-world usefulness. As described earlier, the open-world model is characterized by a user who can visit any web page and an attacker who monitors a subset of those web pages while training a classifier to determine whether the visited web page is in that monitored set. This attack is typically more difficult to carry out, given that the attacker cannot train the model on many of the packet sequences in the unmonitored set.

In this experiment, when the attacker determines that a packet sequence represents a visit to a monitored web page, this prediction is a true positive if correct and a false positive otherwise. Similarly, if the attacker determines that a packet sequence represents a visit to an unmonitored web page, it is a true negative if correct and a false negative otherwise. An attacker is said to have determined that a web page is in the monitored set if the attacker's output probability is above a certain threshold. By varying this threshold, we can calibrate the attacks to achieve high recall or high precision.

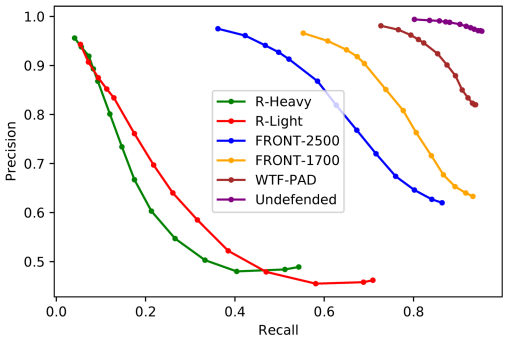



**Fig. 6.** Tik-Tok Precision-Recall

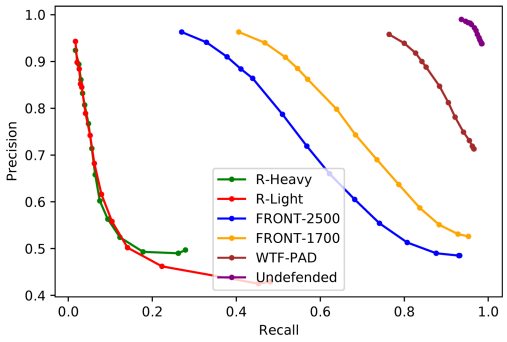



**Fig. 7.** Deep Fingerprinting Precision-Recall

However, true positive rate (TPR) and false positive rate (FPR) are not relevant measures of model performance, since the set of unmonitored web pages may be much larger than the set of monitored ones, causing heavily unbalanced classes. Furthermore, as the unmonitored class size grows, the number of false positives may begin to rival or surpass the number of true positives, making WF attacks impractical [2, 28]. Thus, as described in previous discussions of WF open-world metrics [21, 28, 29, 34, 42], we instead use precision-recall curves to evaluate WF attacks and defenses.

To carry out the open-world experiment, we use the two strongest WF attacks (Tik-Tok and Deep Fingerprinting) and evaluate them against the defenses used in the closed-world setting (except for Tamaraw, which prevented both attacks from reporting more than negligible true positives for many thresholds used). Our training and testing setup models that of the open-world experiment in Deep Fingerprinting [42], which used 85,500 monitored traces (900 each from 95 web pages) with 20,000 unmonitored traces in the training set and 9500 monitored traces (100 each from 95 web pages) with 20,000 unmonitored traces in the testing set.

### 4.2.2 Open-World Results

The precision-recall curves for the Tik-Tok attack against the WF Defenses are shown in Figure 6. As expected, Tik-Tok achieves both high precision and high recall in the undefended data set, demonstrating the effectiveness of the attack in the open-world setting.

WTF-PAD and the FRONT defenses, alternatively, manage to drive down the precision-recall curve, forcing the attacker to choose between high precision with moderate recall and vice versa. However, FRONT-1700 does this more effectively than WTF-PAD, and FRONT-2500 further increases defense performance.

As shown, the RegulaTor defenses outperform their rivals, allowing for high precision only with very low recall. As a result, the effectiveness of the Tik-Tok attack is significantly reduced, as few packet sequences can be determined to be in the monitored set with high reliability.

Against the open-world Deep Fingerprinting attack, the defenses are moderately more effective in terms of driving down precision and recall values, and the RegulaTor settings are about equal in terms of performance. The similarity between the RegulaTor defenses is likely the result of RegulaTor's ability to mask most features based on directional information. Again, RegulaTor shows significantly higher defense performance compared to FRONT, while both FRONT settings show improved performance over WTF-PAD.

To better compare the open-world performance of the tested defenses, consider that the $F_1$-scores of a precision-tuned Tik-Tok are .870, .625, and .135 against WTF-PAD, FRONT-2500, and RegulaTor-Heavy respectively. Note that a lower $F_1$-score implies that the associated defense is more effective. The relatively small $F_1$-score associated with RegulaTor further demonstrates its ability to prevent Tik-Tok from detecting a substantial number of monitored web pages without a high false positive rate as well.

Essentially, the open-world results indicate that the RegulaTor defenses prevent the studied WF attacks from achieving high degrees of precision and recall in a realistic setting. This is made even more apparent by the fact that the open-world setup used in this paper favored the attacker by using a test set where 9500 of the 29,500 traces were from the monitored class. In the

real-world setting, it is unlikely that this proportion of all web pages would be monitored.

## 4.3 Alternate Data Sets

To confirm that the RegulaTor defense generalizes beyond a given data set, we test its performance on two other publicly available website fingerprinting data sets. For simplicity, we only evaluate Regulator-Heavy against the most effective WF attack (Tik-Tok) in the closed-world setting.

First, we test whether the parameters determined from tuning on one data set provide effective defense for other data sets. To do this, we simulated the Regulator-Heavy defense setting on KNN and tested its performance against Tik-Tok. We found that it performed well, reducing Tik-Tok accuracy to 17.8% with 5.1% latency overhead and 77.3% bandwidth overhead. For comparison, Front-2500 defends KNN with a Tik-Tok accuracy of 44.9% and 98.3% bandwidth overhead. In both cases, slightly reduced accuracy is expected given that there are only 90 instances of each website compared to 1000 in DF-CW, limiting the data available to Tik-Tok. Still, it appears that RegulaTor parameters can generalize across data sets, especially given that the data sets were collected a few years apart.

However, it is important to note that the ideal sizes of the packet surges in the RegulaTor defense are dependent on the volume of the undefended traffic sequence. If the RegulaTor surges are too small, then latency will unnecessarily increase as data waits to be sent. While DF-CW and KNN contained similar traffic volume, we may not be able to make this assumption in the real-world setting. To demonstrate this situation, we simulate RegulaTor-Heavy on GE, which contains much higher traffic volume at 5663.9 packets per trace compared to 2100.9 for DF-CW and 1807.6 for KNN. After simulating RegulaTor on GE, we find that RegulaTor-Heavy reduces Tik-Tok to an attack accuracy of 11.3% with 15.1% latency overhead and 39.6% bandwidth overhead. Here, the defense performance is high, but latency overhead is higher as well.

To adjust for the volume of traffic in the data set, we can increase the initial surge rate and padding budget of the defense proportionally to the increased volume of the traffic in the target data set. While this implies that some data collection should be occasionally done to implement the defense, this collection can be fairly minimal, as the adjustment only needs a rough estimate of relative traffic volume.

To demonstrate this adjustment using GE, we multiply the initial surge rate by 2.431, as this represents a relative increase of traffic volume in GE. Additionally, we increase the padding budget by a similar amount. This results in an altered Regulator-Heavy defense with an initial surge rate of 673 and a padding budget of 8030. When simulated on GE, Tik-Tok accuracy is only 5.2%, defense latency overhead is 2.9%, and bandwidth overhead is 82.9%. So, while bandwidth is increased, the altered defense is significantly more effective and operates with reduced latency overhead. For comparison, we test a FRONT defense with increased dummy packet volume to match the FRONT-2500 bandwidth overhead in the original paper [21]. Using a padding budget of 2830 on both the upload and download packets, we find that the FRONT defense reduces Tik-Tok accuracy to 43.4% with a bandwidth overhead of 45.8%. Thus, adjusted RegulaTor is still effective relative to FRONT.

In summary, RegulaTor-Heavy remains effective even when tuned on one data set and then used on another. However, differences in traffic volume between the data set used for tuning and the target data set may cause RegulaTor to incur excess latency or bandwidth overhead. As a result, the initial surge rate and padding budget should be proportionally adjusted as described previously. Then, RegulaTor-Heavy remains highly effective while incurring the intended bandwidth and latency overhead.

## 4.4 Real-world Performance

To obtain more accurate overhead estimates and confirm that RegulaTor could be smoothly used with Tor, we implemented the RegulaTor defense as a pluggable transport [10]. Pluggable Transports (PTs) transform traffic between the client and a bridge in order to disguise the Tor traffic and prevent censorship. Our specific approach was to host a Tor bridge and use the WFPadTools framework [12], which is based on the Obfsproxy pluggable transport [6], to build the RegulaTor defense. Additionally, we used a modified version of tor-browser-crawler [28, 38] to collect the traces for both the RegulaTor-Heavy defense and a 'dummy' transport for comparison. The parameters used in the pluggable transport version of RegulaTor were based on the RegulaTor-Heavy parameters, except that the initial surge rate and download-upload packet ratio parameters were adjusted based on the observed traffic patterns, as described in section 4.3.

| Defenses | Latency OH | Bandwidth OH |
|---|---|---|
| Tamaraw | 36.9% | 196% |
| WTF-PAD | 0% | 54.0% |
| FRONT-1700 | 0% | 81.0% |
| FRONT-2500 | 0% | 119.0% |
| RegulaTor-Light | 8.9% | 48.3% |
| RegulaTor-Heavy | 6.6% | 79.7% |

**Table 4.** Defense Overheads on DF-CW

Then, we used our pluggable transport implementation to collect a RegulaTor-defended data set, PT, consisting of 100 websites and 100 samples per website. To determine the initial surge rate and padding budget, we first collected 10 samples from each of the websites and calculated that the average trace length was 2697.2. Comparing this to the traffic volume found in DF-CW, which was used to tune the original Regulator-Heavy, we then proportionally increased the initial surge rate to 356 and the padding budget to 4564. The remaining parameters were left unchanged, as they appear to generalize regardless of traffic volume.

Using Tik-Tok to test the closed-world WF attack on PT, we record an accuracy of 11.6%. Compared to the traces collected using a dummy pluggable transport, the adjusted RegulaTor-Heavy defense operates with a latency overhead of 13.9% and a bandwidth overhead of 78.2%. While the observed latency overhead appears somewhat higher than our original prediction (as discussed in section 4.5), the adjusted RegulaTor-Heavy defense is more effective as expected. These results demonstrate that RegulaTor can be effective on live traffic and that the parameters found from tuning on a previously collected data set are still valid for a real-world implementation.

## 4.5 Overhead

Table 4 summarizes the bandwidth and estimated latency overheads for each of the tested defenses on DF-CW. The WTF-PAD and FRONT defenses operate with no additional latency, while the RegulaTor defenses incur a small delay on some packet sequences, and the Tamaraw defense causes a substantial delay. Still, since this paper measures latency overhead as the delay of the last 'real' packet, rather than the last dummy packet sent, Tamaraw's latency overhead may appear smaller than in previous works. Here, RegulaTor's latency overhead is an estimate based on the sum of the delay of the last real download packet and the maximum delay of any upload packet. The maximum delay of any upload packet is used because delayed upload requests may delay requests to the web server, delaying download packets even more so.

To illustrate how latency and bandwidth overhead are distributed across various websites, Figure 8 provides the bandwidth and latency multiples as functions of the original trace length, load time, and sending rate for 500 web sites randomly sampled from DF-CW.

As expected, nearly all 'short' traces and traces with a low sending rate have very low latency overhead. This is likely because RegulaTor schedules packets at a much higher rate than the original traces, delaying few packets. However, it is notable that packets with a very short load time often experience high latency overheads as well. This may be because traces with low loading times but high volume send packets at a high rate, which may exceed RegulaTor's sending rate. For the same reason, the sending rate of a trace appears to be correlated with latency overhead. Overall, a majority of traces incur little latency overhead, while traces with a high sending rate more often incur high latency overhead. Also, note that the distribution of latency overhead based on the web page being loaded is particularly important, as increased latency directly harms the user experience.

Additionally, low trace length and sending rate correspond with increased bandwidth overhead. This is expected, as RegulaTor will likely schedule packets at a much faster rate, which results in the frequent sending of dummy packets. Accordingly, traces with high sending rates are associated with low bandwidth overhead ratio. Also, a moderate number of traces with low load time incur high bandwidth overhead. This negative relationship appears to be because traces with low loading time tend to have smaller packet counts, with occasional exceptions.

### 4.5.1 Overhead-performance Trade-offs

To understand the impact of parameter choice, we also simulate RegulaTor on DF-CW with varied parameter values and record the latency overhead, bandwidth overhead, and Tik-Tok performance on the defended data set. To be specific, we used the RegulaTor-Heavy parameters and one by one varied the values of each parameter over a wide range. The results of these experiments are shown in Figure 9.

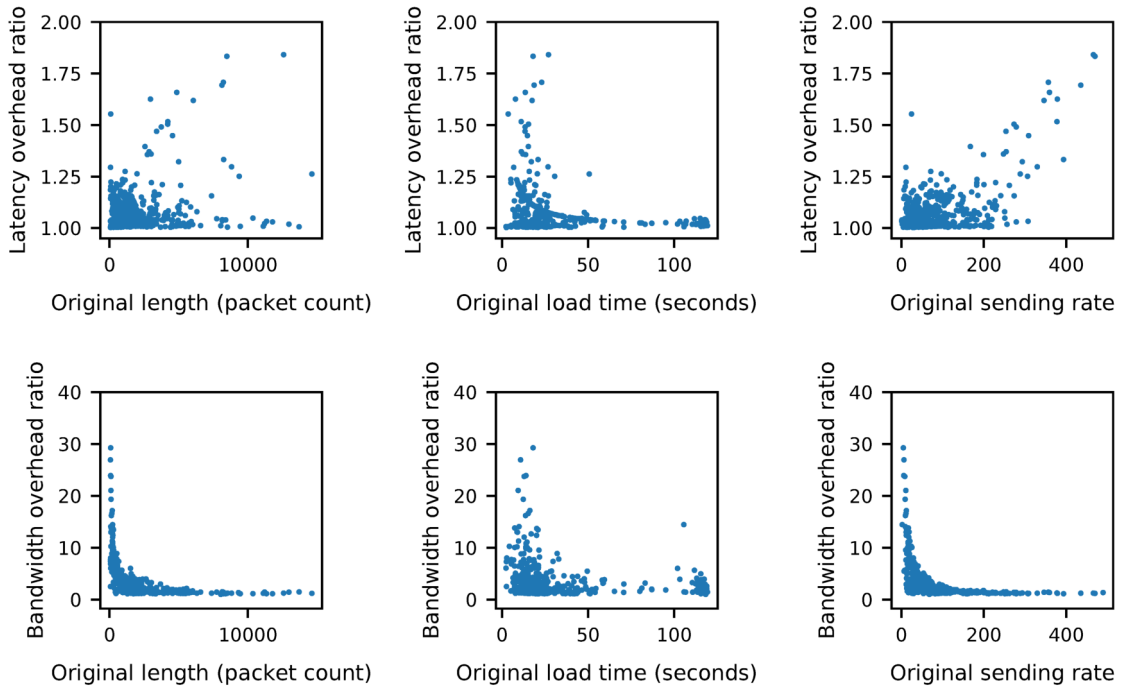



**Fig. 8.** Overhead as a function of trace volume, load time, and packet sending rate

The experiments indicate that RegulaTor performance is fairly stable. Furthermore, varying the parameters allows for RegulaTor to choose different tradeoffs between latency overhead, bandwidth overhead, and defense performance. Specifically, increasing the initial rate increases bandwidth and slightly decreases latency, while increasing the upload ratio decreases the bandwidth but weakens defense performance. Additionally, increasing the padding budget increases bandwidth overhead and defense performance while slightly decreasing latency overhead.

Varying burst threshold and delay cap parameter values had little effect on bandwidth overhead and defense performance, though increasing delay cap increased latency overhead. Lastly, increasing decay rate improved defense performance and decreased latency at the expense of increased bandwidth overhead.

### 4.5.2 Real-world Latency Overhead

While latency overhead can be estimated by simulating RegulaTor on Tor traces, the exact overhead is more difficult to determine, as one would have to accurately predict how the web server would respond to delayed upload packets. However, we can use the latency observed in the real-world data set, which was collected using a PT RegulaTor implementation.

By comparing the loading time between the 'dummy' PT and the RegulaTor PT, we find that the average web page loading time for the RegulaTor defense increased 13.9%. While this exceeds the estimated latency for DF-CW, we find that this is partially explained by differences in traffic patterns.

Specifically, the standard deviation of load time recorded by our dummy PT, at 42.3, is substantially larger than that of DF-CW, which was 28.0. As a result, the real-world RegulaTor implementation more frequently encounters web pages with short load time and high volume or web pages with particularly long loading times. In the former case, RegulaTor will likely delay loading, as it will send packets slower than the high rate in the original trace. In the latter case, RegulaTor will have significantly slowed packet sending near the end of the trace, which may then cause increased latency if a late surge of packets is sent. Even with somewhat increased latency, RegulaTor defense performance in the real-world is strong, reducing Tik-Tok accuracy to 11.6%. If lower latency is desired, then parameters

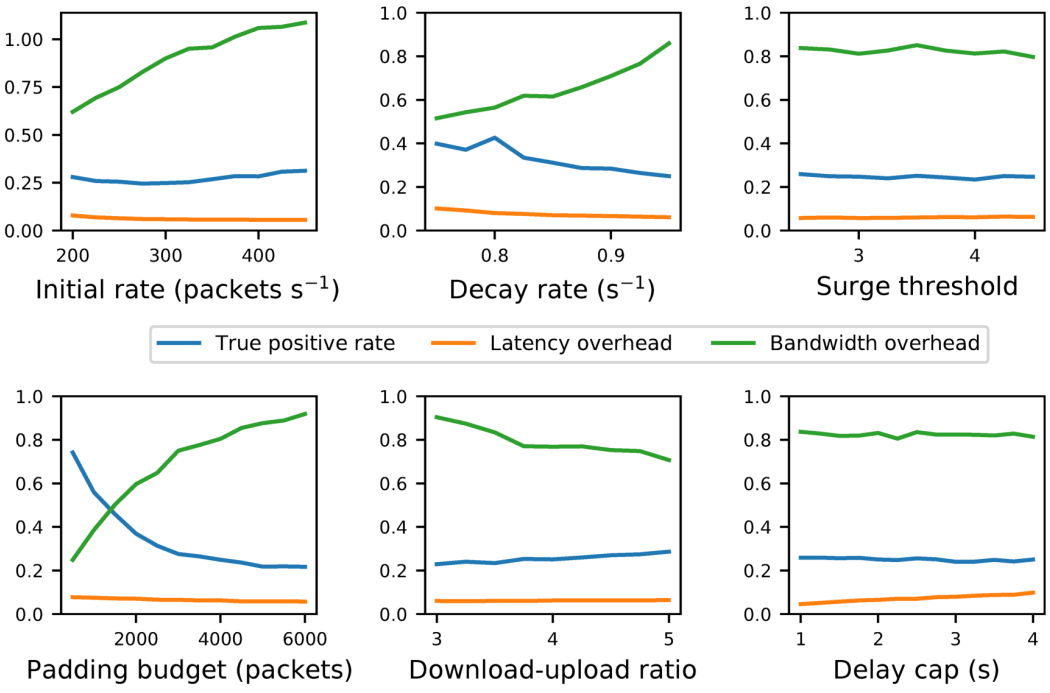



**Fig. 9.** Overhead and closed-world performance by parameter value

may be tuned to decrease it at the expense of a slight bandwidth or performance penalty.

## 4.6 Practicality

Due to RegulaTor's simplicity, it appears that the parameters do not have to be frequently re-tuned over time. Specifically, as long as web traffic can be characterized as predictably 'bursty' (as described in section 3.1), then the surge threshold, delay cap, and decay rate parameters will remain stable. This is further supported by RegulaTor's effectiveness on various data sets with these parameters unchanged. Furthermore, the download-upload packet ratio will remain stable as long as the real-world ratio of download to upload traffic remains constant. While this is difficult to predict, it is notable that both DF-CW and the data collected with the dummy PT, which are collected with similar methodologies but several years apart, report a download-upload packet ratio of 5.96 and 6.66 respectively. Thus, we expect that the download-upload packet ratio parameter will remain somewhat stable.

However, the initial surge rate and padding budget should be increased in response to changes in the average page load bandwidth. This increase appears likely to happen: according to the HTTP Archive [7], the average resource transfer size of tracked URLs increased from 1412.1 KB on July 1, 2017 to 2148.7 KB on July 1, 2021. Accordingly, the ideal initial surge rate and padding budget for the RegulaTor defense may change over time; fortunately, this change is likely to be predictable and straightforward to determine, as one only has to crawl a set of web pages and determine the change in average traffic volume.

In terms of the practicality of a real-world implementation, RegulaTor offers several advantages over previous defenses. First, RegulaTor's overhead is relatively moderate compared to existing regulation defenses, which tend to incur very high latency and bandwidth overhead. RegulaTor also operates without a database of traces associated with other websites, while defenses such as Glove [31], Supersequence [44], and Traffic Morphing [47] require knowledge of other traces. Lastly, RegulaTor does not require additional infrastructure or major changes to the Tor network, while application-level defenses such as HTTPOS [30] require browser modifications and traffic splitting defenses such as TrafficSliver-Net involve major Tor modifications. As a result of these advantages and RegulaTor's computational simplicity, RegulaTor can be straightforwardly implemented as a pluggable transport.

However, a pluggable transport implementation does introduce drawbacks. While PTs protect against the ISP and attackers on the client network, they do not protect against malicious bridges. Moreover, PTs require extra steps for users to set up and use, so many users are likely to remain unprotected. While a circuit padding framework exists [1], allowing developers to inject padding cells between the client and any node within the circuit, it does not allow for real cells to be delayed. Additionally, the circuit padding framework uses adaptive padding-style [11, 29] state machines to describe defenses. As a result, it is unable to support RegulaTor's approach of padding based on the timing of packet surges and buffering packets if necessary. Therefore, updating the circuit padding framework to support this style of defense is a remaining hurdle for the full deployment of RegulaTor.

# 5 Related Work

## 5.1 Website Fingerprinting Attacks

While we use Tik-Tok and Deep Fingerprinting to evaluate WF defenses in this paper, they are not the only deep learning-based techniques that demonstrate high effectiveness. For example, Automated Website Fingerprinting by Rimmer et al. [39] collected a large data set and trained three models: stacked denoising autoencoders, convolutional neural networks, and long short-term memory networks. In addition, Oh et al. presented p-FP [33], which demonstrates that deep neural networks can be used to generate feature vectors to improve previous WF attacks in a variety of settings, and Bhat et al. released Var-CNN [14], which achieves high accuracy while using a reduced data set.

Other works have primarily aimed to improve existing WF attacks. These include Triplet Fingerprinting [43], which uses N-shot learning to allow attackers to train effective WF models on data sets of limited size, and GANDaLF [32], which uses generative adversarial networks to generate "fake" data. Both of these techniques demonstrate that WF attacks can be carried out without the large data sets used in earlier deep learning-based attacks. Lastly, Pulls et al. [37] have shown that the Website Oracle security notion can be combined with website fingerprinting attacks to greatly reduce the false positive rate for most websites visited over Tor.

## 5.2 Website Fingerprinting Defenses

To defend against WF attacks, WF defenses alter traffic to reduce the amount of information leaked about the associated web page. Defenses typically do this by adding 'dummy' packets and inserting delays into the traffic. For clarity, we categorize previously published defenses into four broad categories, which we call 'imitation,' 'regulation,' 'alteration,' and 'traffic splitting.' A summary of published defenses organized by category is given in table 5.

*Imitation* defenses attempt to make packet sequences appear similar or identical to sequences of packets associated with other potential destinations, preventing an attacker from distinguishing between them. While these defenses are often effective with moderate bandwidth overhead, they require information about other packet sequences to imitate. As a result, they require the implementation of additional infrastructure or the storage of data about a user's previously visited web pages, which is potentially difficult. Still, several instances of this defense type have been presented, starting with Traffic Morphing by Wright et al. [47]. This defense used convex optimization techniques to add packets and alter packet lengths to match other packet sequences, which had been previously stored by the client. However, Traffic Morphing is ineffective on Tor, which pads each 'cell' to a constant size. Later, Panchenko et al. [35] presented a method to 'camouflage' a trace by loading another 'decoy' simultaneously with the original one. While this approach appeared relatively effective at first, it has since been defeated by WF attacks.

Another defense, Supersequence, [44] calculates 'anonymity sets,' which are groups of similar packet sequences that are created to minimize the overhead needed to pad them such that an attacker cannot distinguish between them. Then, the shortest common supersequence is calculated to carry out the padding efficiently. The Glove defense [31] uses a similar strategy of clustering web pages using k-medoids and dynamic time warping for distance matrix calculation. Then, it creates 'super-traces' to cover the packet sequences in their respective clusters. While these defenses are theoretically effective, they require knowledge of the full trace being defended and information about packet sequences associated with other web pages.

The most recent imitation defense, Walkie-Talkie, [46], provides a high degree of security with moderate latency and bandwidth penalties. Its approach is to operate Tor Browser in half-duplex mode, meaning that the browser is either loading a page or making re-

| Category | Defense | Data Overhead | Latency Overhead | Additional Requirements | Defeated |
|---|---|---|---|---|---|
| Imitation | Traffic Morphing [47] | Moderate | None | Preliminary Trace Collection | Yes |
| | Decoy pages[35] | High | None | Loads another web page simultaneously | Yes |
| | Supersequence [44] | Moderate | None | Knowledge of other traces | No |
| | Glove [31] | High | High | Knowledge of other traces | No |
| | Walkie-Talkie [46] | Moderate | Moderate | Knowledge of burst information, half-duplex | Yes |
| Regulation | BuFLO [20] | Very High | Very High | None | No |
| | CS-BuFLO [15] | Very High | Very High | None | No |
| | Tamaraw [16] | Very High | Very High | None | No |
| | WTF-PAD [29] | Moderate | None | None | Yes |
| Alteration | HTTPOS [30] | Low | None | Browser modifications | Yes |
| | FRONT [21] | Moderate | None | None | Partially |
| Traffic Splitting | Multihoming [23] | None | None | multipath-compatible bridge | No |
| | TrafficSliver-App [18] | None | Low | Uses local proxy | No |
| | TrafficSliver-Net [18] | None | Low | Tor modifications | No |
| Other | Glue [21] | Moderate | None | User behavior assumptions, trace database | No |

**Table 5.** Summary of WF Defenses

quests, but does not interleave these states. While visiting a web page, the defense uses 'burst molding' to alter the packet bursts to match a packet sequence from another web page. Walkie-Talkie also partially side-steps concerns about knowledge and storage of other packet sequences by storing only burst-level features. Walkie-Talkie is effective against all defenses except for Tik-Tok [38], which uses timing information to distinguish defended packet sequences.

*Regulation* defenses attempt to make WF attacks difficult or impossible by regulating the packet sending. The original regulation defense, BuFLO [20], was primarily created to demonstrate that such a WF defense was possible, albeit inefficient. BuFLO operates by sending packets at fixed intervals for a set length of time. If no packet is available at the set time, then a dummy one is sent instead. Later, CS-BuFLO [15] was presented as an improved version of BuFLO that adapts its transmission rate to reduce overhead and congestion. Another BuFLO variant, Tamaraw [16] reduced overhead while functioning as a "theoretically provable" BuFLO. While these defenses are resistant to WF attacks and require no additional infrastructure or information about other packet sequences, their latency and bandwidth penalties are too high for practical use.

Still, WTF-PAD [29] manages to efficiently regularize some aspects of the packet sequence by using an approach based on adaptive padding [40], which was developed to prevent end-to-end traffic analysis. It does this by filling long gaps in packet sequences whenever the gap between packets is larger than the time length sampled from a distribution of typical inter-arrival times. By reducing the amount of information leaked by each trace while imposing only a moderate bandwidth penalty, WTF-PAD is a strong practical defense; accordingly, WTF-PAD is re-evaluated for comparison in this paper.

*Alteration* defenses aim to change packet sequence features to confuse attackers and make it likely that defended sequences are misclassified. One early example tested in Tor was HTTP pipelining, which consists of multiple HTTP requests sent with a single TCP connection [4]. The HTTPOS defense [30], designed to prevent information leaks from encrypted flows, uses HTTP pipelining along with a variety of other browser changes, such as dummy requests, altered TCP window sizes, multiple TCP connections, and HTTP Range requests. Tor also tested a browser that requested embedded objects randomly. Still, these application-level defenses were defeated by Cai et al. [17].

Another alteration defense is the recently-published FRONT [21], which focuses on adding dummy packets to the front of packet sequences, where most of the information about the associated web page is leaked. It also heavily emphasizes randomized padding by randomly choosing both the distribution and the volume of added packets. As a result, the packet sequences associated with any given web page often look completely different from one another in terms of both total packet volume and location of packet bursts, making it difficult for the WF attacker to accurately categorize packet sequences. FRONT requires no additional latency, a moderate bandwidth penalty, and no data collection or infrastructure. Still, it manages to be one of the most effective of the practical defenses, making it a primary target of comparison in this paper.

*Traffic Splitting* defenses defend against WF attacks by splitting traffic between multiple guard nodes so that any individual sub-trace reveals little information about the target web page. The TrafficSliver [18] and Multihoming [23] defenses both use traffic splitting, though their implementations are fairly different. In Multihoming, clients connect through two different access points (e.g. home WiFi and public WiFi) and then merge traffic at a multipath-compatible Tor bridge. However, TrafficSliver presents two unique approaches known as TrafficSliver-App and TrafficSliver-Net. TrafficSliver-Net alters the Tor network to split TCP traffic over multiple entry nodes and merges traffic at the middle node, while TrafficSliver-App creates multiple Tor circuits and proxies HTTP requests over the circuits.

Both TrafficSliver and Multihoming can be implemented with minimal bandwidth and appear to be effective WF defenses. However, they do not guard against all attacker types: TrafficSliver only prevents WF attacks that take place at the guard node, and Multihoming does not protect against local attackers who can see outgoing traffic. As a result, more investigation into these limitations is needed to further develop traffic splitting attacks.

# 6 Conclusion and Future Work

In this paper, we presented a novel and lightweight WF defense, RegulaTor, along with the insights that allow for it to operate effectively without large bandwidth or latency penalties. By shaping the download packet sequences to contain large packet surges that quickly decay, the packet sequences become difficult to distinguish. Then, because the upload packet sequences closely mimic the download packet sequences, RegulaTor can send upload packets as a function of the download packet timing without incurring high latency or bandwidth overhead. Thus, the download packet sequence is regularized, and the upload packet sequence leaks no further information about the associated web page.

Then, we re-evaluated comparable lightweight WF defenses against state-of-the-art WF attacks to demonstrate that RegulaTor provides substantially improved defense with similar or lessened overhead in both the open-world and closed-world settings. To be specific, it reduces accuracy in the closed-world setting to just 25.4%, compared to 66.0% for FRONT-2500, the best published deployable defense. Furthermore, it achieves this performance with only 79.7% bandwidth overhead and 6.6% latency overhead, while FRONT-2500 requires 119% bandwidth overhead. Most importantly, however, it prevents state-of-the-art WF attacks from detecting visits to monitored web pages with useful accuracy, demonstrated by the precision-tuned Tik-Tok attack receiving an $F_1$-score of only .135 (compared to .625 for FRONT-2500). Thus, RegulaTor represents a step forward in terms of deployable WF defenses.

While RegulaTor prevents an attacker from linking a defended packet sequence with an associated web page, we did not test it in terms of preventing an attack from identifying a web *site* based on a series of page loads. This task may be easier for an attacker able to use information about multiple web pages on the same website as well as information about how users typically interact with these websites. Thus, defending against this type of WF attack is left for future work. Other potentially interesting settings for further analysis include testing RegulaTor with only packet sequences collected using the fastest or slowest circuits, varying the size of the unmonitored or unmonitored sets in the open-world attack, and varying the size of the data set in the closed-world setting.

# Acknowledgments

We'd like to thank the PETS reviewers for their helpful feedback and Sirinam et al. for providing their data sets. This work was funded by a 3M fellowship and NSF grant 1815757.

# References


[1] Tor Project. Circuit padding developer documentation. https://github.com/torproject/tor/blob/master/doc/HACKING/CircuitPaddingDevelopment.md. Accessed: 2020-11-17.

[2] M. Perry. A critique of website traffic fingerprinting attacks. https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks. Accessed: 2020-11-17.

[3] T. Wang. Defenses notes. http://home.cse.ust.hk/~taow/wf/defenses/. Accessed: 2020-9-21.

[4] M. Perry. Experimental defense for website traffic fingerprinting. https://blog.torproject.org/experimental-defense-website-traffic-fingerprinting. Accessed: 2020-11-17.

[5] Hyperopt: Distributed hyperparameter optimization. https://github.com/hyperopt/hyperopt. Accessed: 2021-5-21.

[6] Y. Angel. obfs4 - the obfourscator. https://github.com/Yawning/obfs4. Accessed: 2021-5-21.

[7] HTTP Archive. Report: State of the web. https://httparchive.org/reports/state-of-the-web#bytesTotal. Accessed: 2021-9-21.

[8] Tor Project. Tor browser crawler. https://github.com/webfp/tor-browser-crawler. Accessed: 2020-11-17.

[9] Tor Project. Tor metrics. https://metrics.torproject.org/userstats-relay-country.html. Accessed: 2019-01-20.

[10] Tor Project. Tor: Pluggable transports. https://2019.www.torproject.org/docs/pluggable-transports.html.en. Accessed: 2021-5-21.

[11] Tor Project. Torspec: Padding negotiation. https://gitweb.torproject.org/torspec.git/tree/proposals/254-padding-negotiation.txt. Accessed: 2021-9-21.

[12] M. Juarez. Wfpadtools. https://github.com/mjuarezm/wfpadtools. Accessed: 2021-5-21.

[13] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

[14] S. Bhat, D. Lu, A. Kwon, and S. Devadas. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proceedings on Privacy Enhancing Technologies*, 2019(4):292–310, 2019.

[15] X. Cai, R. Nithyanand, and R. Johnson. Cs-buflo: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 121–130, 2014.

[16] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 227–238. Association for Computing Machinery, nov 2014.

[17] X. Cai, X. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 605–616, New York, NY, USA, 2012.

[18] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko. Trafficsliver: Fighting website fingerprinting attacks with traffic splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1971–1985, New York, NY, USA, 2020.

[19] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symposium (USENIX Security 04)*, San Diego, CA, August 2004. USENIX Association.

[20] K.P. Dyer, S.E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE symposium on security and privacy*, pages 332–346. IEEE, 2012.

[21] J. Gong and T. Wang. Zero-delay lightweight defenses against website fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 717–734. USENIX Association, August 2020.

[22] J. Hayes and G. Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, 2016.

[23] S. Henri, G. Garcia-Aviles, P. Serrano, A. Banchs, and P. Thiran. Protecting against website fingerprinting with multihoming. *Proceedings on Privacy Enhancing Technologies*, 2020:89 – 110, 2020.

[24] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, page 31–42, New York, NY, USA, 2009. Association for Computing Machinery.

[25] A. Hintz. Fingerprinting websites using traffic analysis. In *Proceedings of the 2nd International Conference on Privacy Enhancing Technologies*, PET'02, page 171–178, Berlin, Heidelberg, 2002. Springer-Verlag.

[26] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated Machine Learning - Methods, Systems, Challenges*. 01 2019.

[27] R. Jansen, M. Juarez, R. Galvez, T. Elahi, and C. Diaz. Inside job: Applying traffic analysis to measure tor from within. In *NDSS*, 2018.

[28] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A critical evaluation of website fingerprinting attacks. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 263–274, 2014.

[29] M. Juárez, M. Imani, M. Perry, C. Díaz, and M. Wright. WTF-PAD: Toward an efficient website fingerprinting defense for Tor. *ESORICS 2016*, abs/1512.00524, 2015.

[30] X. Luo, P. Zhou, E.W.W. Chan, W. Lee, R. KC Chang, R. Perdisci, et al. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, volume 11, 2011.

[31] R. Nithyanand, X. Cai, and R. Johnson. Glove: A bespoke website fingerprinting defense. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 131–134. Association for Computing Machinery, nov 2014.

[32] S.E. Oh, N. Mathews, M.S. Rahman, M. Wright, and N. Hopper. Gandalf: Gan for data-limited fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2021(2):305–322, 2021.

[33] S.E. Oh, S. Sunkam, and N. Hopper. p1-FP: Extraction, Classification, and Prediction of Website Fingerprints with Deep Learning. *Proceedings on Privacy Enhancing Technologies*, 2019(3):191–209, 2019.

[34] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[35] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, page 103–114, New York, NY, USA, 2011. Association for Computing Machinery.

[36] T. Pulls. Towards effective and efficient padding machines for tor. *arXiv preprint arXiv:2011.13471*, 2020.

[37] T. Pulls and R. Dahlberg. Website fingerprinting with website oracles. *Proceedings on Privacy Enhancing Technologies*, 2020:235–255, 01 2020.

[38] M.S. Rahman, P. Sirinam, N. Mathews, K.G. Gangadhara, and M. Wright. Tik-tok: The utility of packet timing in website fingerprinting attacks. *Proceedings on Privacy Enhancing Technologies*, 2020(3), 2020.

[39] V. Rimmer, D. Preuveneers, M. Juarez, T.V. Goethem, and W. Joosen. Automated Website Fingerprinting through Deep Learning. In *NDSS*, 2018.

[40] V. Shmatikov and M.H. Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *European Symposium on Research in Computer Security*, pages 18–33. Springer, 2006.

[41] P. Sirinam. Website fingerprinting using deep learning. *Thesis. Rochester Institute of Technology*, 2019.

[42] P. Sirinam, M. Imani, M. Juarez, and M. Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1928–1943, New York, NY, USA, 2018.

[43] P. Sirinam, N. Mathews, M.S. Rahman, and M. Wright. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1131–1148, New York, NY, USA, 2019.

[44] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. *23rd USENIX Security Symposium (USENIX Security 14)*, pages 143–157, 2014.

[45] T. Wang and I. Goldberg. Improved website fingerprinting on Tor. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 201–212, 2013.

[46] T. Wang and I. Goldberg. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, page 1375–1390, USA, 2017. USENIX Association.

[47] C.V. Wright, S.E. Coull, and F. Monrose. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. Technical report, 2009.

# Appendix

## 6.1 Parameter Tuning Details

To find high-performing parameter combinations that achieve specific overhead-performance trade-offs, we use hyperparameter optimization methods. While hyperparameter optimization methods have generally been used for model optimization in machine learning [26], we find that they are fitting for RegulaTor parameter tuning, where parameters are continuous, have set ranges of valid values, and have complex interactions with one another in terms of determining model performance. Furthermore, hyperparameter optimization methods use loss functions to provide feedback about parameter combinations; this allows for RegulaTor to be tuned using a weighted combination of bandwidth overhead, latency overhead, and attack accuracy (which should all be minimized).

There are several classes of approaches to hyperparameter optimization, including grid search, random search, Bayesian optimization, and gradient-based optimization [26]. However, only black-box approaches can be used with RegulaTor, and grid search and random search are likely too time-consuming, as many of the valid combinations of the 6 parameters would have to be tested. On the other hand, Bayesian optimization works by creating a model describing the function being optimized. As more trials are run, the model then provides an increasingly accurate view of which 'regions' of the parameter space are the most promising [13, 26], allowing for more efficient exploration. Of the Bayesian approaches, TPE appears well-established and is supported by the Python library Hyperopt, so we use it to determine the parameters for RegulaTor.

To determine the performance of a given parameter combinations, we use the following loss function for RegulaTor-Heavy:

$$loss = 10l + b + 3a$$

Here, $l$ and $b$ are the latency and bandwidth overhead ratio while $a$ is the accuracy of the leading WF attack, which is Tik-Tok. The loss function for RegulaTor-Light was similar, but more heavily emphasized defense overhead relative to accuracy:

$$loss = 7l + b + 2a$$

After the parameter space was sufficiently explored by the TPE method, the parameter combinations associated with the lowest loss were chosen as RegulaTor-Heavy and RegulaTor-Light.