# *Mockingbird*: Defending Against Deep-Learning-Based Website Fingerprinting Attacks With Adversarial Traces

Mohammad Saidur Rahman, *Graduate Student Member, IEEE*, Mohsen Imani, *Member, IEEE*, Nate Mathews, *Associate Member, IEEE*, and Matthew Wright, *Senior Member, IEEE*

***Abstract*— Website Fingerprinting (WF) is a type of traffic analysis attack that enables a local passive eavesdropper to infer the victim's activity, even when the traffic is protected by a VPN or an anonymity system like Tor. Leveraging a deep-learning classifier, a WF attacker can gain over 98% accuracy on Tor traffic. In this paper, we explore a novel defense, *Mockingbird*, based on the idea of *adversarial examples* that have been shown to undermine machine-learning classifiers in other domains. Since the attacker gets to design and train his attack classifier based on the defense, we first demonstrate that at a straightforward technique for generating adversarial-example based traces fails to protect against an attacker using *adversarial training* for robust classification. We then propose *Mockingbird*, a technique for generating traces that resists adversarial training by moving randomly in the space of viable traces and not following more predictable gradients. The technique drops the accuracy of the state-of-the-art attack hardened with adversarial training from 98% to 42–58% while incurring only 58% bandwidth overhead. The attack accuracy is generally lower than state-of-the-art defenses, and much lower when considering Top-2 accuracy, while incurring lower bandwidth overheads.**

***Index Terms*— Anonymity system, defense, privacy, adversarial machine learning, deep learning.**

## I. Introduction

DEEP learning has had tremendous success in solving complex problems such as image recognition [1], speech recognition [2], and object tracking [3]. Deep learning models are vulnerable, however, to *adversarial examples* – inputs carefully crafted to fool the model [4]. Despite a large body of research attempting to overcome this issue, no methods have been found to reliably classify these inputs. In fact, researchers have found that adversarial examples are another side of the coin of how deep learning models are so successful in the first place [5].

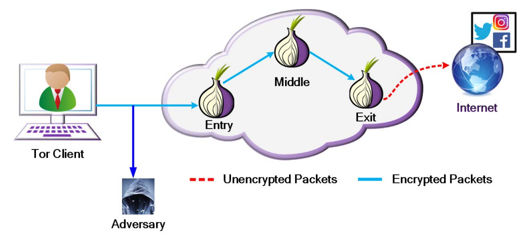



Fig. 1. Website fingerprinting attack model.

In this paper, we investigate whether the exploitability of deep learning models can be used for *good*, defending against an attacker who uses deep learning to subvert privacy protections. In particular, we seek to undermine an attacker using deep learning to perform *Website Fingerprinting* (WF) attacks on the Tor anonymity system.

WF is a class of traffic analysis attack that enables an eavesdropper between the client and the first Tor node on her path to identify which websites the client is visiting. Figure 1 shows the WF attack model. This local passive adversary could be sniffing the client's wireless connection, have compromised her cable/DSL modem, or gotten access to the client's ISP or workplace network.

The WF attack can be modeled as a supervised classification problem, in which the website domain names are labels and each traffic trace is an instance to be classified or used for training. Recently proposed WF attacks [6]–[10] have used deep learning classifiers to great success because of the superior inference capability of deep learning models over traditional machine learning models. The state-of-the-art WF attacks, Deep Fingerprinting (DF) [9] and Var-CNN [8], utilize convolutional neural networks (CNN) to identify patterns in traffic data. These attacks can achieve above 98% accuracy to identify sites using undefended traffic in a *closed-world* setting [8], [9], and both attacks achieve high precision and recall in the more realistic *open-world* setting.

In response to the threat of WF attacks, numerous defenses have been proposed [11]–[13]. WF defenses perturb the traffic so as to hide patterns and confound the classifier. While some defenses have unacceptably high overheads, two relatively lightweight defenses for Tor have recently been proposed:

Manuscript received June 17, 2020; revised September 16, 2020 and October 25, 2020; accepted October 30, 2020. Date of publication November 20, 2020; date of current version December 21, 2020. This work was supported by the National Science Foundation (NSF) under Grant 1423163, Grant 1722743, Grant 1816851, and Grant 1433736. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Wei Yu. *(Mohammad Saidur Rahman and Mohsen Imani contributed equally to this work.) (Corresponding author: Mohammad Saidur Rahman.)*

Mohammad Saidur Rahman, Nate Mathews, and Matthew Wright are with the Global Cybersecurity Institute, Rochester Institute of Technology, Rochester, NY 14623 USA (e-mail: saidur.rahman@mail.rit.edu; nate.mathews@mail.rit.edu; matthew.wright@rit.edu).

Mohsen Imani is with Anomali Inc., Redwood City, CA 94063 USA (e-mail: imani.moh@gmail.com).

This article has supplementary downloadable material available at https://doi.org/10.1109/TIFS.2020.3039691, provided by the authors.

Digital Object Identifier 10.1109/TIFS.2020.3039691

WTF-PAD [14] and Walkie-Talkie (W-T) [15]. State-of-the-art DL attacks, however, have proven effective against both of them [6], [8], [9] and illustrate the need for defenses that can withstand improvements in the attacker's capabilities.

This motivates us to investigate a WF defense that can be effective not only against current DL-based attacks but also against possible attack methods that we can foresee. Adversarial examples are a natural method to turn to for confusing a DL model, so we explore how to create adversarial examples for network traffic. We find that adversarial examples have three attributes that are valuable for defending against WF: i) *high misclassification rates*, ii) *small perturbations*, which ensure a low overhead, and iii) *transferability*. The transferability property [4], in which some adversarial examples can be made to reliably work on multiple models [16], [17], makes it possible to defend against unknown attacks and potentially even ones with more advanced capabilities.

In this paper, we introduce *Mockingbird*,[1] a defense strategy using adversarial examples for network traffic traces, which we call *adversarial traces*. In the WF context, we cannot simply perform a straightforward mapping of adversarial examples to network traffic. As it is the reverse of when the adversary is applying the adversarial examples, the WF attacker gets to know what the WF defender is doing to generate the examples. In particular, he can use the defense as implemented in open-source Tor code to generate his own adversarial traces and then use them to train a more robust classifier. This *adversarial training* approach has been shown to be effective when the classifier knows how the adversarial traces are being generated [18].

To address this, we propose a novel technique to generate adversarial traces that seeks to limit the effectiveness of adversarial training. In particular, we increase the randomness of the search process and reduce the influence of the design and training of the targeted deep learning model in finding a good adversarial example. To this end, as we search for the new trace, we select a random target trace and gradually reduce the distance from the modified trace to the target. We also change to other randomly selected targets multiple times during the search. The deep learning model is only used to give a confidence value on whether the current trace fools the classifier, and we do not access the loss function, the logits, or any other aspect of the model. The resulting adversarial traces can go in many different directions from the original source traces instead of consistently following the same paths that result from, e.g., following the gradient of the loss function. In this way, the technique selects paths that are hard to find through adversarial training. Further, each new trace generated from the same source typically ends near a different target each time, helping to reduce the attacker's Top-$k$ accuracy.

Extensive evaluation shows that *Mockingbird*[2] reliably causes misclassification in a deep learning classifier hardened with adversarial training using moderate amounts of bandwidth overhead. Our results hold even when the attacker uses a significantly more powerful classifier than the target classifier used by *Mockingbird* to produce the adversarial examples.

*Contributions:* In summary, the key contributions of this work are as follows:

- We propose *Mockingbird*, the first WF defense to leverage the concept of adversarial examples.
- We show how algorithms for generating adversarial examples in computer vision fail as a defense in the WF setting, motivating more robust techniques.
- Our evaluation shows that *Mockingbird* significantly reduces accuracy of the state-of-the-art WF attacks hardened with adversarial training from 98% to 38%-58% attack accuracy, depending on the scenario. The bandwidth overhead is 58% for full-duplex traffic, which is better than both W-T and WTF-PAD.
- We show that *Mockingbird* makes it difficult for an attacker to narrow the user's possible sites to a small set. The best attack can get at most 72% Top-2 accuracy against *Mockingbird*, while its Top-2 accuracy on W-T and WTF-PAD is 97% and 95%, respectively.
- Using the WeFDE framework [20], we measure the information leakage of *Mockingbird*, and find that it has less leakage for many types of features than either W-T or WTF-PAD.

Our investigation of this approach provides a promising first step towards leveraging adversarial examples to undermine WF attacks and protect user privacy online.

## II. Threat & Defense Model

### A. Threat Model

We assume that the client browses the Internet through the Tor network to hide her activities (see Figure 1). The adversary of interest is *local*, which means the attacker is positioned somewhere in the network between the client and Tor guard node. The attacker is assumed to already know the identity of the client. His goal is to detect the websites that the client is visiting. A *local* adversary can be an eavesdropper on the user's local network, local system administrators, Internet service provider, any networks between the user and the entry node, or the operator of the entry node. The attacker is *passive*, meaning that the he only observes and records the traffic traces that pass through the network. He does not have the ability to drop, delay, or modify real packets in the traffic stream.

In a website fingerprinting (WF) attack, the attacker feeds the collected network traffic into a trained machine-learning or deep-learning classifier. For the purpose of training, the WF attacker first needs to collect traffic of various sites by operating a Tor client. Since is not feasible to collect traffic for all the sites on the web, the attacker identifies a set of *monitored* sites that he wants to track. The attacker limits the scope of his attack to the identification of any website visits that are within the monitored set. The set of all other sites is known as the *unmonitored* set.

WF attacks and defenses are evaluated in two different settings: *closed-world* and *open-world*. In the closed-world setting, we assume that the client is limited to visiting only

[1]The Northern mockingbird imitates the calls of a wide range of other birds, and one of its call types is known as a *chatburst* that it uses for territorial defense: https://en.wikipedia.org/wiki/Northern_mockingbird

[2]This work is an extended version of a short paper [19].

the *monitored* sites. The training and testing set used by the attacker only include samples from the monitored set. The closed-world scenario models an ideal setting for the attacker and is not indicative of the attack's real-world performance. From the perspective of developing a WF defense, demonstrating the ability to prevent closed-world attacks is thus sufficient to show its effectiveness.

In contrast, the open-world scenario models a more realistic setting in which the client may visit websites from both the monitored and unmonitored sites. In this setting, the attacker trains on the monitored sites and a representative (but not comprehensive) sample of unmonitored sites. The open-world classifier is then evaluated against both monitored and unmonitored sites, where the set of unmonitored sites used for testing does not intersect with the training set.

### B. Defense Model

The purpose of a WF defense is to prevent an attacker who observes a traffic trace from determining accurately to which site the trace belongs. To achieve this, the real traffic stream must be manipulated in some way. Because traffic is bidirectional, the deployment of a successful WF defense requires participation from both the client and a cooperating node in the Tor circuit. We call this node the *bridge*.[3] To defend against eavesdroppers performing WF attacks, the *bridge* could be any node located between the adversary and the client's destination server, making it so that the adversary only has access to the obfuscated traffic stream. Since the guard node knows the IP address of the client and can thus act as as a WF adversary, it is better to set up the bridge at the middle node, which cannot directly identify the client.

## III. Background & Related Work

### A. WF Attacks

Website fingerprinting attacks have applied a variety of classifiers. The three best attacks based on manual feature engineering are $k$-NN [21], CUMUL [22], and $k$-FP [23], which all reached over 90% accuracy in closed-world tests on datasets with 100 samples per site. In the rest of this section, we examine the more recent deep-learning-based WF attacks.

*1) SDAE:* The first to investigate using deep-learning techniques for WF were Abe and Goto [24], who developed an attack based on Stacked Denoising Autoencoders (SDAE). Their model was trained on raw packet direction, represented by a sequence of "$+1$" and"$-1$" values for outgoing and incoming packets, respectively. Despite this innovation, their attack achieved a lower accuracy rate than the previous state-of-the-art attacks at only 88% in the closed-world setting.

*2) Automated Website Fingerprinting (AWF):* Rimmer *et al.* [10] proposed using deep learning to bypass the feature engineering phase of traditional WF attacks. To more effectively utilize DL techniques, they collected a very large dataset of 900 sites with 2,500 trace instances per site. They applied several different DL architectures—SDAE, Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM)—on the traffic traces. They found that their CNN model outperforms the other DL models they developed, obtaining 96% accuracy in the closed-world setting.

*3) Deep Fingerprinting (DF):* Sirinam *et al.* [9] developed a deeper CNN model that reached up to 98% accuracy rate in the closed-world setting using a dataset of 100 sites with 1,000 instances each. They also examined the effectiveness of their model against WF defenses, where they showed that their model can achieve concerningly high performance against even some defended traffic. Most notably, their attack achieved 90% accuracy against WTF-PAD [14] and 98% Top-2 accuracy against Walkie-Talkie [15].

*4) Var-CNN:* Recently, Bhat *et al.* [8] developed a more sophisticated WF attack based on the ResNet CNN architecture and attained 98.8% closed-world accuracy.

We evaluated *Mockingbird* on both the DF model and Var-CNN model in the black-box setting.

### B. WF Defenses

To defeat WF attackers, researchers have explored various defense designs that generate cover traffic to hide the features present in website traffic. WF defenses are able to manipulate the traffic stream with two operations: sending dummy packets and delaying real packets. These manipulations, however, come at a cost: sending dummy packets adds an additional bandwidth overhead to the network, while delaying packets adds latency overhead that directly impacts the time required to load the page. Several studies have thus tried to balance the trade-off between the WF defense's overhead and efficacy of the defense against WF attacks. In this section, we review these WF defenses.

*1) Constant-Rate Padding Defenses:* This family of defenses transmits traffic at a constant rate in order to normalize trace characteristics. BuFLO [25] is the first defense of this kind, and it sends the packets in the same constant rate in both directions. The defense ends transmission after the page has finished loading and a minimum amount of time has passed. The overhead of the traffic is governed by both the transmission rate and the minimum time threshold for the stopping condition. Moreover, although the defense covers fine-grained features like burst information, course-grained features like the volume and load time of the page still leak information about the website. Tamaraw [13] and CS-BuFLO [12] extend the BuFLO design with the goal of addressing these issues. To provide better cover traffic, after the page is loaded, Tamaraw keeps padding until the total number of transmitted bytes is a multiple of a fixed parameter. Similarly, CS-BuFLO pads the traffic to a power of two, or to a multiple of the power of the amount of transmitted bytes. All of these defenses are expensive, requiring two to three times as much time as Tor to fetch a typical site and more than 100% bandwidth overhead.

*2) Supersequence Defenses:* This family of defenses depends on finding a *supersequence* for traffic traces. To do this, these defenses first cluster websites into anonymity sets and then find a representative sequence for each cluster, such that it contains all the traffic sequences. All the websites that

[3]Tor bridges are usually used for evading censorship, but they can be used for prototyping WF defenses such as used in WTF-PAD [14].

belong to the same cluster are then molded to the representative supersequence. This family includes Supersequence [21], Glove [11], and Walkie-Talkie [15]. Supersequence and Glove use approximation algorithms to estimate the supersequence of a set of sites. The traces are then padded in such a way so as to be equivalent to its supersequence. However, applying the molding directly to the cell sequences creates high bandwidth and latency costs. Walkie-Talkie (WT) differs from the other two defenses in that it uses anonymity sets of just two sites, and traces are represented as burst sequences rather than cell sequences. Even with anonymity sets of sizes of just two, this produces a theoretical maximum accuracy of 50%. Wang and Goldberg report just 31% bandwidth overhead for their defense, but also 34% latency overhead due to the use of half-duplex communication. Against WT, the DF attack achieved 49.7% accuracy and 98.4% *top-2* accuracy, meaning that it could effectively identify the two sites that were molded together but not distinguish between them [9].

*3) WTF-PAD:* Shmatikov and Wang [26] proposed *Adaptive Padding (AP)* as a countermeasure against end-to-end traffic analysis. Juarez *et al.* [14] proposed the WTF-PAD defense as an adaptation of AP to protect Tor traffic against WF attacks. WTF-PAD tries to fill in large delays between packets *(inter-packet arrival times)*. Whenever there is a large inter-packet arrival time (where "large" is determined probabilistically), WTF-PAD sends a fake burst of dummy packets. This approach does not add any artificial delays to the traffic. Juarez *et al.* show that WTF-PAD can drop the accuracy of the $k$-NN attack from 92% to 17% with a cost of 60% bandwidth overhead. Sirinam *et al.* [9], however, show that their DF attack can achieve up to 90% accuracy against WTF-PAD in the closed-world setting.

*4) Application-Level Defenses:* Cherubin *et al.* [27] propose the first WF defenses designed to work at the application layer. They proposed two defenses in their work. The first of these defenses, ALPaCA, operates on the webserver of destination websites. ALPaCA works by altering the size distribution for each content type, e.g. PNG, HTML, CSS, to match the profile for an average onion site. In the best case, this defense has 41% latency overhead and 44% bandwidth overhead and reduces the accuracy of the CUMUL attack from 56% to 33%. Their second defense, LLaMA, operates exclusively on the client. It adds random delays to HTTP requests in an effort to affect the order of the packets by manipulating HTTP request and responses patterns. LLaMA drops the accuracy of the CUMUL attack on Onion Sites from 56% to 34% at cost of 9% latency overhead and 7% bandwidth overhead.

## IV. Preliminaries

### A. Adversarial Examples

Szegedy *et al.* [4] were the first to discover that otherwise accurate ML and DL image classification models could be fooled by image inputs with slight perturbations that are largely imperceptible to humans. These perturbed inputs are called *adversarial examples*, and they call into question the robustness of many of the advances being made in machine learning. The state-of-the-art DL models can be fooled into misclassifying adversarial examples with surprisingly high confidence. For example, Papernot *et al.* [17] show that adversarial images cause a targeted deep neural network to misclassify 84% of the time.

The idea of creating adversarial examples is to modify samples from one class to make them be misclassified to another class, where the extent of the modification is limited. More precisely, given an input sample $x$ and target class $t$ that is different from actual class of $x$ ($t \neq C^*(x)$), the goal is to find $x'$ which is close to $x$ according to some distance metric and $C(x') = t$. In this case, $x'$ is a *targeted* adversarial example since it is misclassified to a particular target label $t$. An *untargeted* adversarial example, on the other hand, may be misclassified to any other class except the true class ($C^*(x)$).

In response to the threat of adversarial examples, many defense techniques have been introduced to make classifiers more robust against being fooled. Recent research [28], [29] shows that almost none of these recent defense techniques are effective. In particular, we can generate adversarial examples that counter these defense techniques by including the defense techniques directly into the optimization algorithm used to create the adversarial examples. We can also overcome many defense approaches by simply increasing the amount of perturbation used [30].

### B. Properties of Adversarial Examples

Adversarial examples have three major properties that make them intriguing for us in WF defense: i) robust misclassification, ii) small perturbations, and iii) transferability. We now explain the effect of each of these properties in a WF defense.

*1) Robust Misclassification:* An effective defense should be able to fool a trained WF classifier consistently in real-world conditions. Adversarial examples have been shown to work reliably and robustly for images, including for cases in which the viewpoint of the camera cannot be fully predicted, such as fooling face recognition [31] and self-driving cars [32].

*2) Small Perturbations:* To fool the classifier, the defense will add padding packets to the original network traffic. Ideally, a WF defense should be lightweight, meaning that the number of padding packets should be constrained to keep bandwidth consumption low. By using small perturbations to achieve misclassification, an effective WF defense based on adversarial examples can also be lightweight.

*3) Transferability:* Recent research shows that defenses such as WTF-PAD [14] and W-T [15], which defeated the state-of-the-art attacks available at that time, are seriously underminded by the more recent and advanced attacks [6], [9]. Given that the attacker could use any classifier for WF, the defense should extend beyond current attacks to other possible attacks as well. Adversarial examples provide the ability for this due the the *transferability* property, which indicates that they can be designed to attack a given classifier and at least sometimes also fool other classifiers [16]. Further, there are techniques that work in a *black-box* setting, where the classifier is completely unknown to the attacker [17], [33]. This property is very important for WF defense, since we cannot predict the attacker's classifier in advance.

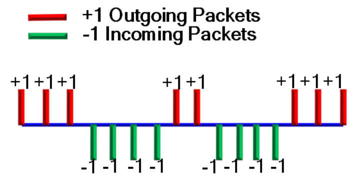



Fig. 2. A visual representation of bursts.

### C. Adversarial Training

Recent research shows that adversarial training increases the robustness of a model by incorporating adversarial examples in the training data [18], [34]. The idea is to train a network with adversarial examples so that they can be classified correctly. This approach is limited, as it does not adapt well to techniques for generating adversarial examples that haven't been trained on. In the WF setting, however, the classifier has the advantage of knowing how the adversarial examples are being generated, as they would be part of the open-source Tor code. Thus, adversarial training is a significant concern for our system.

### D. Data Representation

Following the previous work [9], [14], [15], we model the traffic trace as a sequence of incoming (server to client) and outgoing (client to server) bursts. The incoming and outgoing packets are represented as $-1$ and $+1$, respectively. We define a burst as a sequence of consecutive packets in the same direction. An example of the burst representation is shown in Figure 2. Given this, we can increase the length of any burst by sending padding packets in the direction of the burst. We cannot, however, decrease the size of the bursts by dropping real packets due to the retransmissions this would cause, changing the traffic patterns and adding delays for the user.

## V. *Mockingbird* DESIGN

We now motivate and describe the design of the *Mockingbird* defense in detail. We start by evaluating the performance given by adapting existing adversarial-example techniques, which are not effective against adversarial training, and then examine the *Mockingbird* design in detail.

### A. Applying Existing Methods in WF Defense

Several different algorithms have been proposed for generating adversarial examples within the field of computer vision, including the Fast Gradient Sign Method (FGSM) [35], the Iterative Fast Gradient Sign Method (IGSM) [36], the Jacobian-Based Saliency Map Attack (JSMA) [37], and optimization-based methods [29], [33]. For our initial exploration of adversarial examples in WF defense, we examine the technique proposed by Carlini and Wagner (C&W) [16].

This method is shown to defeat the defensive distillation approach of blocking adversarial examples [38]. The algorithm is successful with 100% probability. We modified their technique to suit our needs to generate adversarial traces out of the burst sequences. This algorithm is designed to work on images, which are 2D, so we modified it to work on 1D traffic traces.

We evaluated the performance of this technique in two different WF attack scenarios: *without-adversarial-training* and *with-adversarial-training*. The without-adversarial-training scenario represents the scenario most typically seen in the adversarial example literature, in which the classifier has not been trained on any adversarial instances. In this scenario, we generated the adversarial examples against a target model and tested them against the different WF attacks trained on the original traffic traces. We find that our adversarial traces are highly effective against the WF attacks. The accuracy of DF [9] is reduced from 98% to 3%, and the accuracy of CUMUL [22] drops from 92% to 31%. The adversarial traces generated using this method are highly transferable, as we generated them against a target CNN model similar to the one proposed by Rimmer *et al.* [10], and they are effective against both DF and CUMUL.

Unfortunately, this scenario is not realistic, as it is likely (and usually assumed) that the attacker can discern what type of defense is in effect and train on representative samples. This is represented by the *with-adversarial-training* scenario. In this scenario, the C&W technique fails completely, with the DF attack reaching 97% accuracy. In addition, we also investigated a method that combines aspects of our *Mockingbird* system with C&W, but this also proved to be ineffective as a WF defense. We discuss the details of these evaluations in the *Supplemental Materials*.

The results of this evaluation led to a very important insight: the scenario in which the effectiveness of adversarial examples are typically evaluated is notably different than that of a WF defense. In particular, the attacker has the advantage of going second, which means that the classifier can be designed and trained after the technique is deployed in Tor. Thus, techniques that excel at producing adversarial examples for traditional attacks are poorly suited for our problem. In response to this discovery, we focused our efforts on the development of a new technique designed specifically for our needs. We discuss our method in the following section.

### B. Generating Adversarial Traces

We now introduce *Mockingbird*, a novel algorithm to generate adversarial traces that more reliably fool the classifier in the adversarial training setting. The ultimate goal is to generate untargeted adversarial traces that cause the classifier to label a traffic trace as coming from some site other than the original site, i.e. to generate an untargeted sample. We find, however, that it is more effective to generate targeted samples, i.e. to select specific other sites for the current sample to attempt to mimic. Much like its namesake (the bird), *Mockingbird* uses a variety of targets, without much importance on which target it mimics at any given moment.

To defend a given trace, the *source sample*, *Mockingbird* first generates a set of potential target traces selected randomly from the traces of various sites other than the source site. It then randomly picks one of these traces as the *target sample* and gradually changes the source sample to get closer

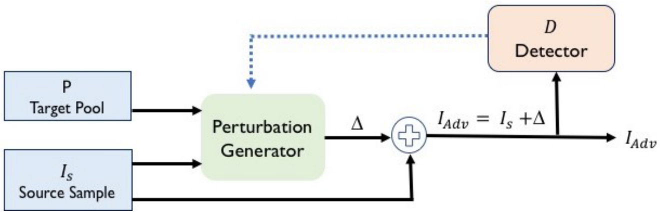



Fig. 3. *Mockingbird* architecture.

to the target sample. The process stops when a trained classifier called the *detector* determines that the class of the sample has changed (see Figure 3). Note that it does not need to have changed to the target sample's class, as the goal is to generate an untargeted adversarial trace. The amount of change applied to the source sample governs the bandwidth overhead of *Mockingbird*, and as such should be minimized.

Unlike most other algorithms used to generate adversarial examples, *Mockingbird* does not focus on the loss function (like FGSM and IGSM) or logits (like Carlini & Wagner) of the detector network. Instead, it aims to move the source sample towards the target sample and only uses the detector network to estimate the confidence with which the trace is misclassified. This lessens the reliance on the shape of the detector network and helps to generate adversarial traces that are more robust against adversarial training. As we demonstrate in the *Supplementary Materials*, using an optimization method to move towards the target sample results in a system that is much less robust.

Mockingbird *Algorithm:* We assume that we have a set of sensitive sites $\mathcal{S}$ that we want to protect. We train a *detector* $f(x)$ on a set of data from $\mathcal{S}$. We discuss the design and training of $f(x)$ in Section VI. We consider traffic trace $I_s$ as an instance of source class $s \in \mathcal{S}$. Our goal is to alter $I_s$ to become $I'_s$ such that it is classified to any other class $t$, $t = f(I'_s)$ and $t \neq s$.

$I_s$ is a sequence of the bursts, $I_s = \left[b_1^I, b_2^I, \ldots, b_n^I\right]$, where $n$ is the number of the bursts. Burst is defined as the sequence of packets in a direction (i.e. incoming or outgoing) [8], [9], [21]. The length of each burst (i.e. number of packets in each burst), $b_i^I$, makes the sequence of bursts. Usually, the number of packets in each burst vary widely. The only allowed operation on a burst $b_i^I$ is to add some positive values $\delta_i >= 0$ to that burst, $b_i^I = b_i^I + \delta_i$. The reason for using $\delta_i >= 0$ is that we can only increase the size of a burst. If $\delta_i < 0$, that would mean we should drop some packets to reduce the size of a burst, and dropping real packets means losing data and requires re-transmission of the dropped packet. To protect source sample $I_s$, we first select $\tau$ potential target samples from other classes $t_i \neq s$. We then select the target $t$ as the one nearest to the source sample based on the $l_2$ norm distance.[4] This helps to minimize overhead, as we will move the source towards the target. More formally, we pick a *target pool* $P_s$ of $p$ random samples from other classes, $P_s = \left[I^0{}_0, I^1{}_1, .., I^p{}_m\right]$, where $I^j{}_i$ is the $j$-th sample in the target pool and belongs to target class $t_i \neq s$. The target sample $I_t$ is selected as shown in Equation 1.

$$I_t = \underset{I \in P_s}{\operatorname{argmin}}\ D(I_s, I) \tag{1}$$

$$D(x, y) = l_2(x - y) \tag{2}$$

To make the source sample leave the source class, we change it with the minimum amount of perturbation in the direction that makes it closer to the target ($I_t$). We define $\Delta$ as the perturbation vector that we add to the source sample to generate its defended form $I_s^{new}$.

$$\Delta = [\delta_0, \delta_1, \cdots, \delta_n] \quad (\delta_i >= 0) \tag{3}$$

$$I_s^{new} = I_s + \Delta \tag{4}$$

We need to find a $\Delta$ that adds the least amount of perturbation to the source sample while still making it closer to the target sample. Therefore, we find $\Delta$ that minimizes distance $D(I_s^{new}, I_T)$. To do so, we compute the gradient of the *distance* with respect to the input. Note that most work in adversarial example generation uses the gradient of the loss function of the discriminator network rather than distance, and this may make those techniques more sensitive to the design and training of the classifier. The gradient points in the direction of steepest ascent, which would maximize the distance. Therefore, we compute the gradient of the negative of the distance with respect to the input, and we modify the source sample in that direction towards the target sample. In particular:

$$\nabla(-D(I, I_T)) = -\frac{\partial D(I, I_T)}{\partial I} = \left[-\frac{\partial D(I, I_T)}{\partial b_i}\right]_{i \in [0, \cdots, n]} \tag{5}$$

where $b_i$ is the i-th burst in input $I$.

To modify the source sample, we change bursts such that their corresponding values in $\nabla(-D(I, I_T))$ are positive. Our perturbation vector $\Delta$ is:

$$\delta_i = \begin{cases} -\alpha \times \dfrac{\partial D(I, I_T)}{\partial b_i} & -\dfrac{\partial D(I, I_T)}{\partial b_i} > 0 \\ 0 & -\dfrac{\partial D(I, I_T)}{\partial b_i} \leqslant 0 \end{cases} \tag{6}$$

where $\alpha$ is the parameter that amplifies the output of the gradient. The choice of $\alpha$ has an impact on the convergence and the bandwidth overhead. If we pick a large value for $\alpha$, we will take bigger steps toward the target sample and add more overhead, while small values of $\alpha$ require more iterations to converge. We modify the source sample by summing it with $\Delta$, ($I_s^{new} = I_s + \Delta$). We iterate this process, computing $\Delta$ for $I_s$ and updating the source sample at each iteration until we leave the source class, $f(I_s^{new}) \neq s$ or the number of iterations passes the maximum allowed iterations. Note that at the end of each iteration, we update the current source sample with the modified one, $I_s = I_s^{new}$. Leaving the source class means that we have less confidence on the source class. So we fix a threshold value, $\tau_c$, for measuring the confidence. If the confidence of the *detector* on the source class is less than the threshold ($f_s(I_s^{new}) < \tau_c$), *Mockingbird* will stop changing the source sample ($I_s$).

[4]We also performed some preliminary experiments with *Manhattan* distance and found no significant improvement.

As we only increase the size of the bursts where $-\frac{\partial D(I, I_T)}{\partial b_i} > 0$, we may run into cases that after some iterations $\nabla(-D(I, I_T))$ does not have any positive values or all the positive values are extremely small such that they do not make any significant changes to $I_s$. In such cases, if $I_s^{new} - I_s$ is smaller than a threshold $\tau_D$ for $\lambda$ consecutive iterations (we used $\lambda = 10$), and we are still in the source class, we select a new target. In particular, we effectively restart the algorithm by picking a new pool of potential target samples, selecting the nearest target from the pool, and continuing the process. It is to note that, the source sample at this point is already in the changed form $I_s^{new}$ and the algorithm starts changing from $I_s^{new}$. In this process, the confusion added by *Mockingbird* in the final trace is proportional to the number of targets changed to reach the final adversarial trace. The pseudo code of *Mockingbird* algorithm is presented in Algorithm 1.

**Algorithm 1** Generate Adversarial Traces

**Input** : $\mathcal{S}$ – set of sensitive sites
$\mathcal{D}$ – detector $f(x)$
$\alpha$ – amplifier
$\delta$ – iterations before selecting a new target
$\tau_c$ – confidence threshold
$\tau_D$ – perturbation threshold
$N$ – maximum number of iterations
$p$ – number of targets to pick for the pool
$I_s$ – instance of site $s$ to protect
$b_j$ – bursts in $I_s$, $j \in [1, \ldots, n]$
**Output**: $I_s'$ – altered (adversarial) trace of $I_s$

$Y_s \leftarrow \mathcal{D}(I_s)$ ; `// label of` $I_s$
$P_s \leftarrow random(p, S - \{s\})$ ; `// target pool`
$I_T \leftarrow \operatorname{argmin}_{I \in P_s} D(I_s, I)$ ; `// target sample;`
$I_s' \leftarrow I_s$
**for** *iter in* $[1, \ldots, N]$ **do**
  $\nabla(-D(I_s', I_T)) \leftarrow \left[-\frac{\partial D(I_s', I_T)}{\partial b_j}\right]_{j \in [1, \cdots, n]}$
  $\Delta \leftarrow \alpha \times \nabla(-D(I_s', I_T))$, where $-\frac{\partial D(I_s', I_T)}{\partial b_j} > 0$
  $I_s' \leftarrow I_s' + \Delta$
  ; `/* Compute the label and confidence for` $I_s'$ `*/`
  $Y_s', \; P(I_s') \leftarrow \mathcal{D}(I_s')$
  ; `/* End if the source class confidence is low */`
  **if** $Y_s' \neq Y_s$ ***and*** $P(I_s') < \tau_c$ **then**
    break;
  ; `/* Pick a new target after` $\delta$ `iterations */`
  **if** $i \bmod \delta = 0$ ***and*** $Y_s' = Y_s$ ***and*** $I_s' - I_s < \tau_D$ **then**
    $P_s \leftarrow random(p, S - \{s\})$ ; `// new target pool`
    $I_T \leftarrow \operatorname{argmin}_{I \in P_s} D(I_s, I)$ ; `// new target sample`
**return** $I_s'$

## VI. Evaluation

### A. Datasets

We apply *Mockingbird* to generate adversarial examples on the traffic traces at the burst level. We can get the burst sequence of the traffic traces from both full-duplex (FD) and half-duplex (HD) communication modes. Walkie-Talkie (W-T) [15] works on HD communication, and it finds the supersequences on the burst level. In our experiments, we use burst sequences for both FD and HD datasets.

*1) Data Source:* We use both the closed-world (CW) and open-world (OW) FD and HD traffic traces provided by Sirinam *et al.* [9]. The websites classes from the monitored set are from the top sites in Alexa [39].

*2) Preprocessing Data:* In our preprocessing phase, we filter the data by removing any instances with fewer than 50 packets and the instances that start with an incoming packet, since the client should send the first packet to start the connection.

*3) Full-Duplex (FD):* The CW dataset contains 95 classes with 1000 instances each. After preprocessing, we end up with 518 instances for each site. The OW dataset contains 40,716 different sites with 1 instance each.

*4) Half-Duplex (HD):* The CW dataset contains 100 sites with 900 instances each. After preprocessing, we ended up with 83 classes with 720 instances per class. The OW data contains 40,000 sites with 1 instance each.

An additional consideration is that we must use a fixed size input to our model [9]. To find the appropriate size, we consider the distribution of burst sequence lengths within our datasets. Figure 4 shows the CDF of the burst sequence lengths for both the HD and FD datasets. More than 80% of traces have fewer than 750 bursts for the HD CW dataset, and more than 80% of the traces for the CW FD dataset have fewer than 500 bursts. We found that using 1500 bursts on both HD and FD datasets provides just 1% improvement on accuracy for the DF attack compared to using 750 bursts. To decrease the computational cost for generating adversarial examples, we use an input size of 750 bursts for both the FD and HD datasets. Note that the attacker in our evaluations uses 10,000 packets rather than bursts.

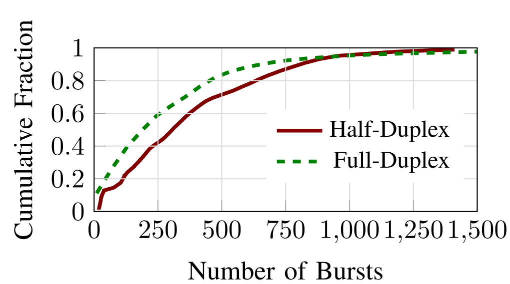



Fig. 4. CDF of the number of bursts in the full-duplex and half-duplex traces.

In our evaluation, we need to address the needs of both *Mockingbird* and the adversary to have training data. We thus break each dataset (full-duplex (FD) and half-duplex (HD)) into two non-overlapping sets: *Adv Set* $\mathcal{A}$ and *Detector Set*

TABLE I

DATASET SPLIT: *Adv Set* ($\mathcal{A}$) & *Detector Set* ($\mathcal{D}$). **FD**: FULL-DUPLEX, **HD**: HALF-DUPLEX, **C**: CLASS, **I**: INSTANCE, **CW**:CLOSED-WORLD, **OW**: OPEN-WORLD

| | *Adv Set* $\mathcal{A}$ ($C \times I$) | *Detector Set* $\mathcal{D}$ ($C \times I$) | CW Total | OW |
|---|---|---|---|---|
| FD | 95×259 | 95×259 | 95×518 | 40,716 |
| HD | 83×360 | 83×360 | 83×720 | 40,000 |

$\mathcal{D}$ (see Table I). $\mathcal{A}$ and $\mathcal{D}$ each contain half of the instances of each class. This means, in both $\mathcal{A}$ and $\mathcal{D}$, 259 instances for each of 95 classes for FD data and 360 instances for each of 83 classes for HD data.

## B. Experimental Method

*Mockingbird* needs to train a detector $f(x)$ with instances of a variety of sites to generate adversarial traces. We use both Rimmer *et al.*'s CNN model [10], which we refer to as AWF, and Sirinam *et al.*'s more powerful DF model [9] for detector models. We train these models on the traces of the Detector Set. Sirinam *et al.* suggest using an input size of 5,000 packets, but our padded traces have more traffic, so we use an input size of 10,000 packets, which is the 80th percentile of packet sequence lengths in our defended traces. Sample from the Detector Set are *only* used to train the detector. The source samples from Adv Set ($I_s \in \mathcal{A}$) are used to generate adversarial traces for the training, validation, and testing of the adversary's classifier. To evaluate the defended traffic, we reform the defended traces from the burst level representation to the direction level representation where "+1" and "−1" indicate outgoing and incoming packets, respectively, following the previous research.

*1) Attack Settings:* We test with two different settings, white-box and black-box. In all of the evaluations, the classifier is trained using adversarial training, where the attacker has full access to the defense and uses it to generate defended samples for each class in the monitored set.

- *White-box*: In the white-box setting, we assume that that defense uses the same network architecture for the detector as the attacker uses to perform WF attacks. We use this overly optimistic scenario only for parameter search to identify values for $\alpha$ and number of iterations, where we use DF as both the detector and the attack classifier.
- *Black-box*: In the black-box setting, the defender and the attacker use two different neural networks, i.e. the defender uses one model for the detector, while the attacker uses another model for performing WF attacks. We evaluate *Mockingbird* in the black-box setting by using the AWF CNN model [10] as the detector and both the DF model [9] and Var-CNN [8] as attacker models. Since DF and Var-CNN are more powerful than the simple AWF model [8], [9], this tests the case that the attacker has greater capabilities than are known to the defender. These evaluations show the extent to which adversarial examples generated by *Mockingbird* transfer to other classifiers.
- *Traditional ML Attack*: We also evaluate *Mockingbird* against tradition machine-learning (ML) attacks such as CUMUL [22], *k*-FP [23], and *k*-NN [21], all of which reach over 90% accuracy on undefended Tor using closed-world datasets with 100 samples per class.

TABLE II

HYPERPARAMETER TUNING ON DF AND VAR-CNN ATTACK MODELS FOR BLACK-BOX ATTACKS

| **Hyperparameters** | **Choices** | **Final** DF | **Final** Var-CNN |
|---|---|---|---|
| Training Epoch | [30, 50, 100, 150] | 100 | 100 |
| Batch Size | [32, 64, 128, 256] | 32 | 32 |
| Optimizer | [Adam, Adamax] | Adamax | Adamax |
| Learning Rate | [0.001, 0.002, 0.003] | 0.002 | 0.002 |
| Activation Fn. | [ReLU, ELU] | ReLU, ELU | ReLU |

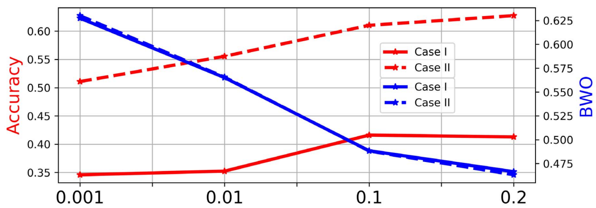
*2) Training and Hyperparameter Tuning:* To perform the attacks, we use 80% of the data for training, 10% for validation, and the remaining 10% for testing for each of the settings. To represent the data in the attack models, we follow the prior work [8], [9], [21]–[23] and represent the data as an 1-D vector of +1 and −1 for outgoing and incoming packets, respectively. We use a fixed length of 5000 for each instance of the class following the prior work. Instances that do not have 5,000 packets are padded with zero, and the instances that have length more than 5,000 are truncated to that particular length.

As black-box setting is the most realistic attack setting to evaluate a defense, we perform hyperparameter tuning on the two deep-learning based attack models: DF [9] and Var-CNN [8]. However, we exclude the hyperparameters choices that have been explored in prior work and did not provide any improvement in the attack accuracy. Such hyperparameters include the SGD and RMSPros optimization functions, and the tanh activation function.

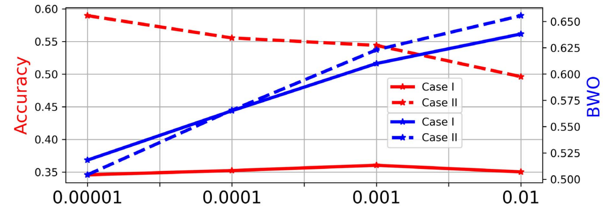
We start our search with the default model and continue tuning each hyperparameter one by one. First, we perform several sets of experiments with different training epochs for both full-duplex (FD) and half-duplex (HD) with both DF [9] and Var-CNN [8] models. Then, the best training epoch number is used for the experiments to search for the next set of hyperparameters. Table II shows the hyperparameters and the choices of our hyperparameter tuning process. In our tuning process, we found that the tuned DF and Var-CNN models work better for the FD traces defended by *Mockingbird* and default DF and Var-CNN models work better for the HD traces defended by *Mockingbird*.

*3) Top-k Accuracy:* Most prior works have focused their analysis on Top-1 accuracy, which is normally referred to simply as the accuracy of the attack. We argue that Top-1 accuracy does not provide a full picture of the effectiveness of a defense, as an attacker may use additional insights about their target (language, locale, interests, etc.) to further deduce what website their target is likely to visit.

As such, it is desirable to examine the accuracy of Top-*k* predictions, where *k* is the number of top-ranked classes in the prediction. In evaluations of WF, we are particularly interested in the Top-2 accuracy. A high Top-2 accuracy indicates that the classifier is able to reliably determine the identity of a trace to just two candidate websites. This is a threat to a user even when the attacker is unable to use additional information




(a) Confidence Threshold Value ($\tau$ )




(b) Perturbation Threshold Value ($\tau_D$)

Fig. 5. *Full-duplex*: the attacker's accuracy and the bandwidth overhead (BWO) of the generated samples as we vary the probability threshold value (Figure 5a) and perturbation threshold value (Figure 5b).

to predict the true site. The knowledge that the target *may* be visiting a sensitive site is actionable and can encourage the attacker to expend additional resources to further analyze their target.

*4) Target Pool: Mockingbird* changes the source sample toward a target sample drawn randomly from our target pool. The detector determines whether the perturbed source sample is still in the source class. We are interested to know how the algorithm performs if we fill the target pool with instances of sites that the detector has been trained on and has not been trained on. We examine the bandwidth overhead and reduction in the attack accuracy of traces protected by *Mockingbird* in these two scenarios.

- *Case I*: We fill the target pool with instances from the Adv Set. Therefore, both source samples ($I_s \in \mathcal{A}$) and target samples (${I^j}_i \in \mathcal{A}$ which $T_i \neq s$) are from the Adv Set. In this case, we assume that the detector has been trained on the target classes, which makes it more effective at identifying when the sample has left one class for another. This may be less effective, however, if the adversary trains on the source class but none of the other target classes used in detection.
- *Case II*: We fill the target pool with instances from unmonitored sites that are not in the Adv Set. We select the target samples (${I^j}_i$) from the open-world dataset. The source samples ($I_s$) are from Adv Set, and we generate their defended forms. In this case, we assume the detector has not been trained on the target samples, so it may be less effective in identifying when the sample leaves the class. That may make it more robust when the attacker also trains on a different set of classes in his monitored set.

Case I may be realistic when the defender and the attacker are both concerned with the same set of sites, such as protecting against a government-level censor with known patterns of blocking, e.g. based on sites it blocks for connections not protected by Tor. Case II is a more conservative estimate of the security of *Mockingbird* and thus more appropriate for our overall assessment.

We generate defended samples with various settings. We vary $\alpha$ to evaluate its effect on the strength of the defended traces and the overhead. We also vary the number of iterations required to generate the adversarial traces. Each iteration moves the sample closer to the target, improving the likelihood it is misclassified, but also adds bandwidth overhead.

### C. Tuning on Full-Duplex Data

The full-duplex version of *Mockingbird* is the easier to deploy and leads to lower latency costs than the half-duplex version [15], so we lead with the full-duplex results. We use the white-box setting for simplicity.

*1) Choice of Threshold Values:* There are two thresholds values in *Mockingbird* algorithm: i) a confidence threshold value ($\tau_c$) that limits the confidence on the generated trace belonging to the correct class, and ii) a perturbation threshold value ($\tau_D$) that limits the amount of change in a generated trace. Intuitively, the smaller the value of $\tau_c$, the less likely it is that the attacker could succeed, at the cost of higher average bandwidth overhead. By contrast, changing $\tau_D$ directly manipulates the maximum bandwidth overhead, where lower $\tau_D$ would reduce bandwidth cost but also typically result in an improved chance of attack success.

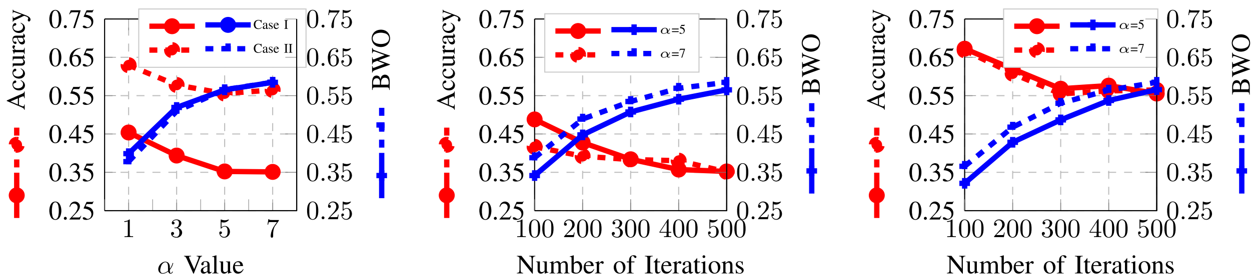
While we experiment with different $\tau_c$ values, we fix the $\tau_D = 0.0001$, $\alpha = 5$, and number of iterations to 500. We can see from Figure 5a that lower values of $\tau_c$ lead to lower accuracy and higher bandwidth. $\tau_c = 0.01$ provides a good tradeoff point, so we select it for our following experiments.

Following the same procedure, we perform experiments with different $\tau_D$ values. As expected, and as shown in Figure 5b, lower $\tau_D$ values lead to lower bandwidth but higher attack accuracy. We choose $\tau_D = 0.0001$ as a good tradeoff for our following experiments.

*2) Choice of $\alpha$:* Figure 6a shows the bandwidth overhead and attack accuracy of full-duplex data with respect to $\alpha$ values for both Case I (solid lines) and Case II (dashed lines) with 500 iterations. As expected, the bandwidth overhead increases and the attack accuracy decreases as we increase $\alpha$, with longer steps towards the selected targets. For Case I, the adversary's accuracy against *Mockingbird* with $\alpha = 5$ and $\alpha = 7$ are both 35%, but the bandwidth overhead is lower for $\alpha = 5$ at 56% compared to 59% for $\alpha = 7$. For Case II, the adversary's accuracy and the bandwidth overhead are both slightly lower for $\alpha = 5$ than that of $\alpha = 7$. From these findings, we fix $\alpha = 5$ for our experiments.

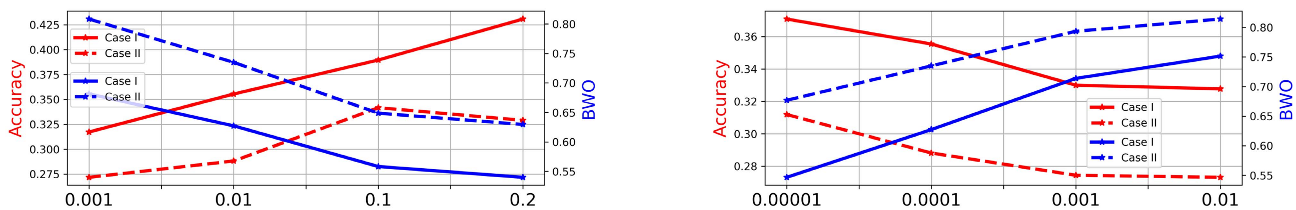
We also observe that Case I leads to lower accuracy and comparable bandwidth overhead to Case II. When $\alpha = 5$ and $\alpha = 7$, the attack accuracies for Case I are at least 20% lower than that of Case II. Therefore, as expected, picking target samples from classes that the detector has been trained on drops the attacker's accuracy.

*3) Number of Iterations:* Figure 6 shows the trade-off between the accuracy and bandwidth overhead with respect




(a) $\alpha$ Value for full-duplex (b) Case I (c) Case II

Fig. 6. *Full-duplex*: the attacker's accuracy and the bandwidth overhead (BWO) of the generated samples as we vary the $\alpha$ value (Figure 6a) and number of iterations (Figure 6b and 6c).




(a) Confidence Threshold Value ($\tau_c$) (b) Perturbation Threshold Value ($\tau_D$)

Fig. 7. *Half-duplex*: the attacker's accuracy and the bandwidth overhead (BWO) of the generated samples as we vary the probability threshold value (Figure 7a) and perturbation threshold value (Figure 7b).

to the number of iterations to generate the adversarial traces. As mentioned earlier, increasing the number of iterations also increases the number of packets (overhead) in the defended traces. We vary the number of iterations from 100 to 500 for both Case I (Figure 6b) and Case II (Figure 6c) to see their impact on the overhead and the accuracy rate of the DF attack.

For Case I, we can see that the DF attack accuracy for both 400 and 500 iterations is 35% when $\alpha = 5$, while the bandwidth overheads are 54% and 56%, respectively. For $\alpha = 7$, the attacker's accuracy is higher and the bandwidth costs are higher. For Case II, using $\alpha = 5$ leads to 57% accuracy with 53% bandwidth overhead for 400 iterations and 55% accuracy and 56% bandwidth overhead for 500 iterations. From these findings, we fix the number of iterations to 500 for our experiments.

### *D. Tuning on Half-Duplex Data*

Using half-duplex communication increases the complexity of deployment and adds latency overhead [15], but it offers more precise control over burst patterns such that we can achieve reduced attacker accuracy as shown in the following white-box results.

*1) Choice of Threshold Values:* We can see from Figure 7a and 7b that $\tau_c = 0.01$ and $\tau_D = 0.0001$ provide better trade-off between attack accuracy and bandwidth overhead for HD data as well. The attack accuracies are 35% and 29%, and bandwidth overheads are 73% and 63% for case I and case II, respectively. Hence, we select $\tau_c = 0.01$ and $\tau_D = 0.0001$ for our next set of experiments.

*2) Choice of $\alpha$:* As seen in Figure 8 (all for 500 iterations), the lowest accuracy rates are 35.5% and 28.8% for Case I and Case II, respectively, when $\alpha = 7$. The bandwidth overheads are 62.7% and 73.5% for Case I and Case II, respectively. When $\alpha = 5$, the attack accuracy is 50% for both Case I and Case II with bandwidth overheads of 57% and 69%, respectively. As expected, higher $\alpha$ values mean lower attacker accuracies at the cost of higher bandwidth. Additionally, as in the full-duplex setting, both bandwidth overhead and accuracies are lower in Case I than Case II. From these findings, we set $\alpha = 7$ for our experiments.

*3) Number of Iterations:* Figure 9 shows the trade-off between attacker accuracy and bandwidth overhead with the number of iterations for $\alpha = 7$. We vary the number of iterations from 100 to 500 for both Case I and Case II. For Case I, the accuracy is 35.5% with 62.7% bandwidth overhead with 500 iterations. With 400 iterations, the accuracy is 37% with 59% bandwidth overhead. For Case II, with 500 iterations, we can get the lowest attack accuracy of 28.8%, with a cost of 73.5% bandwidth overhead. From these findings, we set the number of iterations to 500 for our experiments.

### *E. Results Analysis*

We lead our analysis with bandwidth overhead followed by the analysis of white-box setting, black-box setting, and traditional ML attacks. We extend our analysis with a discussion of Top-$k$ accuracy. The analysis is based on the best parameters ($\tau_c$, $\tau_D$, $\alpha$ and number of iterations) found in Section VI-C and VI-D where $\tau_c = 0.01$, $\tau_D = 0.0001$, and $\alpha = 5$ and $\alpha = 7$ for FD and HD datasets, respectively. The number of iterations are 500 for both datasets. In addition, we include the investigation of multiple-round attacks on *Mockingbird*. To compare *Mockingbird* with other defenses, we selected two state-of-the-art lightweight defenses: WTF-PAD [14] and W-T [15]. To generate defended traffic,

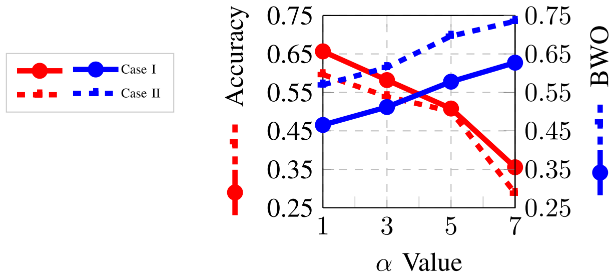



Fig. 8. *Half-duplex*: attacker accuracy and bandwidth overhead (BWO) as $\alpha$ varies.

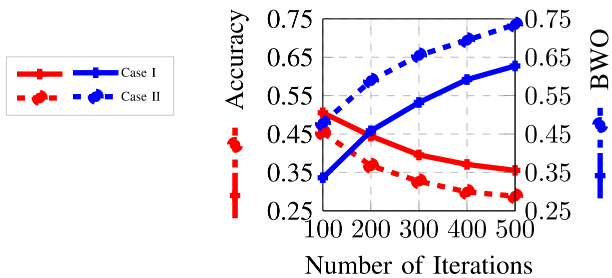



Fig. 9. *Half-duplex*: attacker accuracy and bandwidth overhead (BWO) as the number of iterations varies.

we simulated WTF-PAD on our FD dataset, as it works with full-duplex traffic, while we simulated W-T on our HD datasets, since W-T uses half-duplex traffic. Thus, in our experiments, the WTF-PAD dataset has 95 sites with 518 instances each and W-T has 82 sites with 360 instances–note that we use just the *Adv Set* $\mathcal{A}$ from the HD dataset for a fair comparison with *Mockingbird*.

For our W-T simulation, we used half-duplex data from Sirinam *et al.* [9].[5] There are modest differences between this data and the data used by Wang and Goldberg [15]. Perhaps most importantly, Sirinam *et al.*'s implementation uses a newer version of Tor. Also, Sirinam *et al.*'s dataset is from websites in 2016 rather than 2013, which we have observed makes a significant difference in their distributions. These differences likely account for the differences in reported bandwidth in our findings and those of Wang and Goldberg.

*1) Bandwidth Overhead:* As described in Section V-B, we designed *Mockingbird* to minimize packet addition, and thus bandwidth overhead, by keeping the amount of change in a trace under a threshold value of 0.0001. From Table IV, we can see that for full-duplex (FD) network traffic, the bandwidth overhead in Case I and Case II of *Mockingbird* are the same at 58%, which is 6% and 14% lower than WTF-PAD and W-T, respectively. For half-duplex (HD) *Mockingbird*, the bandwidth overhead is 62% for Case I and 70% for Case II.

*2) White-Box Setting:* In the white-box setting, shown in Table III, *Mockingbird* is highly effective at fooling the attacker. In Case I, the attacker gets less than 40% accuracy for both FD and HD traffic. For Case II, the attacker actually has a lower accuracy of 29% in the HD setting, but reaches 55% in the FD setting.

[5]Tao Wang's website (http://home.cse.ust.hk/~taow/wf/, accessed Sep. 10, 2020) mentions that Sirinam *et al.*'s W-T browser, which is designed to collect half-duplex data, is a better implementation.

TABLE III

**WHITE-BOX.** EVALUATION OF *Mockingbird* AGAINST DF. **BWO**: BANDWIDTH OVERHEAD, **FD**: FULL-DUPLEX, **HD**: HALF-DUPLEX

| Cases | Dataset | BWO | DF [9] |
|---|---|---|---|
| **Case I** | FD | 0.56 | 0.35 |
| | HD | 0.63 | 0.35 |
| **Case II** | FD | 0.56 | 0.55 |
| | HD | 0.73 | 0.29 |

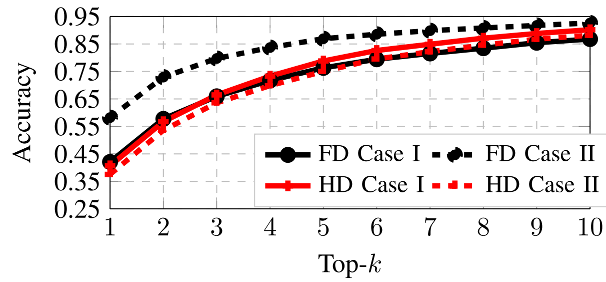



Fig. 10. **Black-Box Top-$k$ Accuracy.** DF accuracy for different values of $k$ against *Mockingbird*.

*3) Black-Box Top-1 Accuracy:* In the more realistic black-box setting, our results as shown in Table IV indicate that *Mockingbird* is an effective defense. For Case I and Case II, the respective attack accuracies are at most 42% and 62%. In comparison, Var-CNN achieves 90% closed-world accuracy against WTF-PAD and 44% against W-T. So the effectiveness of *Mockingbird* in terms of Top-1 accuracy falls between these two defenses.

*4) Traditional ML Attacks:* As shown in Table IV, the highest performance for any of CUMUL, $k$-FP, and $k$-NN against *Mockingbird* was 32% (Case II, Full Duplex). *Mockingbird* is competitive with or better than both WTF-PAD and W-T for all three attacks. This shows that the effectiveness of *Mockingbird* is not limited to protecting against DL models, but also against traditional ML.

*5) Top-k Accuracy:* Our results in Table IV show that *Mockingbird* is somewhat resistant to Top-2 identification, with an accuracy of 72% in the worst case. On the other hand, W-T struggles in this scenario with 97% Top-2 accuracy by DF, as its defense design only seeks to provide confusion between two classes. In addition, Var-CNN attains 95% Top-2 accuracy against WTF-PAD. This Top-2 accuracy of *Mockingbird* indicates a notably lower risk of de-anonymization for Tor users than WTF-PAD or W-T.

In addition to Top-2, we analyzed the Top-10 accuracy of DF against *Mockingbird*. Figure 10 shows that *Mockingbird* can limit the Top-10 accuracies of full-duplex (FD) data to 87% and 92% for Case I and Case II, respectively. For half-duplex (HD), Top-10 accuracies are 90% and 88% for Case I and Case II. Overall, the worst-case Top-10 accuracy is about the same as the Top-2 accuracy against WTF-PAD, while the worst-case Top-10 accuracy is significantly better than the Top-2 accuracy for W-T.

## F. Intersection Attacks

In this section, we evaluate the effectiveness of *Mockingbird* in a scenario in which the adversary assumes that the user

TABLE IV

**Black-Box.** AWF CNN AS DETECTOR. EVALUATION OF *Mockingbird* AGAINST DF, VAR-CNN, CUMUL, $k$-FP, AND $k$-NN ATTACKS & COMPARISON AGAINST WTF-PAD AND W-T DEFENSES. **BWO**: BANDWIDTH OVERHEAD, **FD**: FULL-DUPLEX, **HD**: HALF-DUPLEX

| Case | Dataset | BWO | DF [9] | Var-CNN [8] | CUMUL [22] | $k$-FP [23] | $k$-NN [21] | DF Top-2 | Var-CNN Top-2 |
|---|---|---|---|---|---|---|---|---|---|
| | Undefended (FD) | - | 0.97 | 0.98 | 0.93 | 0.85 | 0.86 | - | - |
| | Undefended (HD) | - | 0.98 | 0.99 | 0.92 | 0.92 | 0.90 | - | - |
| | WTF-PAD [14] | 0.64 | 0.86 | 0.90 | 0.55 | 0.44 | 0.17 | 0.92 | 0.95 |
| | W-T [15] | 0.72 | 0.40 | 0.44 | 0.36 | 0.30 | 0.35 | 0.97 | 0.94 |
| **Case I** | *Mockingbird* (FD) | 0.58 | 0.42 | 0.35 | 0.19 | 0.21 | 0.08 | 0.56 | 0.50 |
| | *Mockingbird* (HD) | 0.62 | 0.41 | 0.33 | 0.22 | 0.28 | 0.10 | 0.57 | 0.47 |
| **Case II** | *Mockingbird* (FD) | 0.58 | 0.58 | 0.62 | 0.32 | 0.32 | 0.14 | 0.70 | 0.72 |
| | *Mockingbird* (HD) | 0.70 | 0.38 | 0.30 | 0.20 | 0.26 | 0.12 | 0.54 | 0.43 |

is going to the same site regularly (e.g. every day), and that the adversary is in a position to monitor the user's connection over multiple visits. Both of these assumptions are stronger than in a typical WF attack, but they are common in the literature on attacking anonymity systems, such as *predecessor attacks* [40], [41] and *statistical disclosure attacks* [42], [43]. In this setting, the attacker can leverage weaker information such as top-10 classification results to eventually deanonymize a user who uses Tor to visit the same site every day.

While more sophisticated statistical methods could be adopted [42], [43], we have limited data for this purpose and instead use a simple *intersection attack* to shed light on the information available to the attacker. In an intersection attack, the adversary examines the top-$k$ results of a WF attack applied to each of the user's Tor connections. Treating each result as a set of $k$ sites, he takes the intersection of the sets to eliminate sites that do not appear every day. If the results consistently identify a single site as being in the top-$k$ results, then one may speculate that this site is in fact where the user is going online using Tor every day.

If the WF defense is very consistent, in that the same $k$ sites are in the top-$k$ results every day, then this kind of intersection attack will not reduce the security of the defense over time. In that sense, W-T provides an assurance that the top-2 accuracy holds for this stronger attack model.

For *Mockingbird*, however, there is no direct assurance. We thus test the intersection attack over a period of five rounds (i.e. five days). We model multiple rounds by randomly selecting a set of test instances with the same label from our dataset. For each round, we compute the top-10 predicted labels ($T_{10}$) and take the intersection of that set with the intersected labels ($L_{int}$) computed in the previous round. Mathematically, this attack process can be expressed as $\nabla L_{int}^{n} = T_{10}^{n} \cap L_{int}^{n-1}$ for each attack round $n > 1$.

The metrics we use for evaluating *Mockingbird* in this scenario are: i) *absolute success*, ii) *absolute failure*, and iii) *mean intersection*. After five rounds, if $L_{int}$ consists of only the correct class, we call this an absolute success. Absolute failure is the case where $L_{int}$ is empty, or the correct class is not in $L_{int}$. Finally, for cases where $L_{int}$ contains two or more classes including the correct class, we take the average size of $L_{int}$ as the mean intersection.

We use the tuned DF model for *Mockingbird* FD data and the default DF model for *Mockingbird* HD data for these experiments. The results are shown in Table V. Absolute successes for FD are 27% and 36% for case I and case II, respectively; they are below 20% for both HD cases. The absolute failure rates are slightly above 50% for FD case I and both HD cases, though just 20% for FD case II. Mean intersections for FD are 2.86 and 2.60 for case I and case II, respectively, and 4.12 for both HD cases.

TABLE V

**INTERSECTION ATTACK.** EVALUATION OF *Mockingbird* AGAINST AN INTERSECTION ATTACK OVER FIVE ROUNDS. **FD**: FULL-DUPLEX, **HD**: HALF-DUPLEX

| Cases | Dataset | Absolute Success | Absolute Failure | Mean Intersection |
|---|---|---|---|---|
| **I** | *Mockingbird* (FD) | 0.27 | 0.51 | 2.86 |
| | *Mockingbird* (HD) | 0.17 | 0.52 | 4.12 |
| **II** | *Mockingbird* (FD) | 0.36 | 0.20 | 2.60 |
| | *Mockingbird* (HD) | 0.13 | 0.53 | 4.12 |

From this, we see that the intersection attack is only moderately helpful for the attacker against *Mockingbird*. Even when the attacker can assume that the user visits the same site regularly, the top-10 labels do not include the real site in a significant fraction of cases. Further, there are often multiple sites in the intersection set. Thus, while *Mockingbird* does not provide assurance against attacks over multiple rounds of observation by a determined attacker, it is also not highly vulnerable to such an attack.

### G. Information Leakage Analysis

Recent works have argued that classification accuracy is not a complete metric to evaluate the objective efficacy of a WF defense [20], [44]. We thus adopted the WeFDE information leakage estimation technique [20] to evaluate *Mockingbird*. WeFDE allows us to determine how many bits of information each feature leaks for a given defense. The authors of the WeFDE paper made their code available to us; for speed and memory requirements, we use the re-implemented version of the code from Rahman *et al.* [6]. We perform information leakage analysis on undefended full-duplex traffic and defended traffic for 3043 manually defined features spread across 14 categories. The defenses we examine are WTF-PAD, W-T, and the full-duplex Case I variant of *Mockingbird*. The leakage for each feature and defense is shown in Figure 11.

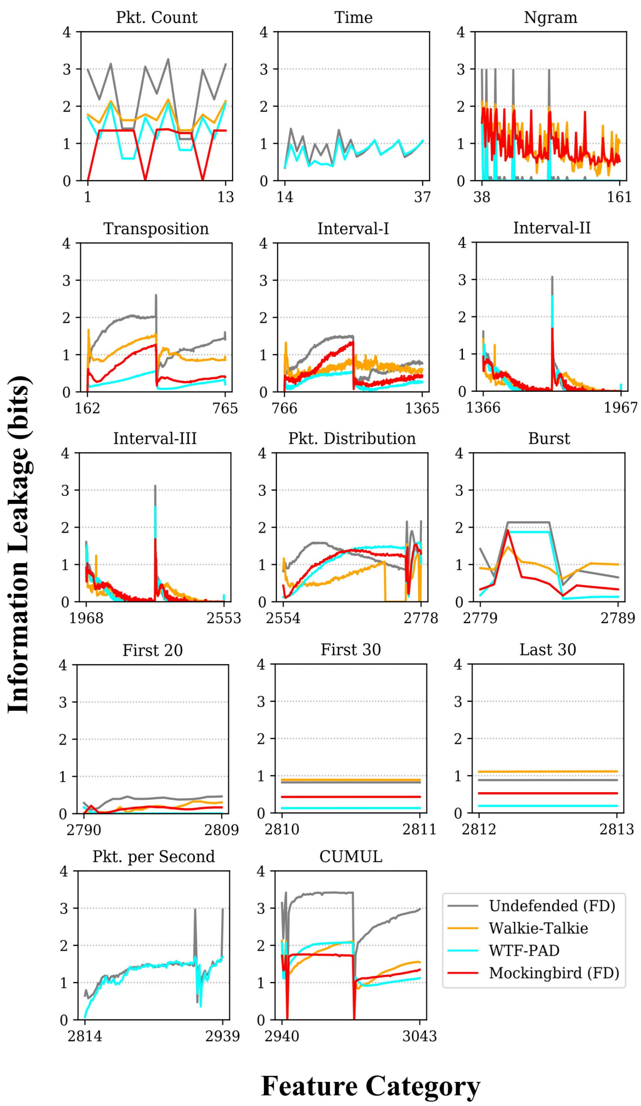



Fig. 11. Individual feature information leakage.

Our *Timing* and *Pkt. per Second* categories do not include W-T or *Mockingbird* measurements, as the simulations for these defenses are unable to produce accurate timestamp estimations.

In general, we find that the information leakage of *Mockingbird* to be comparable to the other defenses. We find that any given feature leaks at most 1.9 bits of information in *Mockingbird*. W-T has similar maximum leakage, with at most 2.2 bits per feature, while WTF-PAD leaks at most 2.6 bits. The maximum amount of leakage seen for the undefended traffic was 3.4 bits, nearly twice that of *Mockingbird*. *Mockingbird* seems to be most vulnerable to the *N*-gram class of features, which is not surprising, as these features seem to be effective for all traffic defenses we examined. On the other hand, it is quite effective against packet count features, which was previously the most significant class of features for the undefended traffic and also useful against W-T. Additionally, *Mockingbird* shows notable improvements over W-T in the *Transposition*, *First 30*, *Last 30*, and *Burst* features, while W-T is better than *Mockingbird* in the *Pkt. Distribution* feature.

Overall, the results of our information leakage analysis are largely consistent with what we see in our accuracy measurements.

## VII. Discussion

### A. Comparison

*Mockingbird* has significant advantages over the state-of-the-art defenses. Compared to WTF-PAD, the Top-1 accuracies of DF and Var-CNN are at least 28% lower, effectively tripling the error rate for the attacker. Compared to W-T, the Top-1 accuracies of DF and Var-CNN are relatively higher, but the Top-2 accuracies are at least 27% and 22% lower, respectively. We believe that the Top-2 accuracy of 97% for W-T should be considered unacceptable given the costs of deploying and running the defense.

One question that our findings raise is *Why does* Mockingbird *perform so well in Top-k accuracy?* We hypothesize that the random selection of targets during the search for an adversarial example is helping to create unpredictable patterns that not only move away from the original site class but move towards a number of different possible classes. W-T is explicitly designed to provide confusion only among a limited number of sites, two by default. WTF-PAD has random patterns, but these may only lead to confusion among sites that are already similar. *Mockingbird* can find new sites that confuse the attacker.

Note that our black-box models assume that we use a detector model (AWF) that is weaker than state-of-the-art models (DF and Var-CNN). Employing a more powerful detector model could thus have the potential to make *Mockingbird* robust against even more powerful models that may be developed in the future.

Unlike W-T, *Mockingbird* can be deployed with full-duplex communication and thus lower latency overhead [15]. Finally, *Mockingbird*'s bandwidth overhead in full-duplex mode is modestly better than both of these defenses.

### B. Implementation & Deployment

To deploy *Mockingbird* in the real world, it is necessary to address several outstanding issues.

*1) Live Trace Generation:* We have yet to find a solution that allows for live packet-by-packet generation of adversarial traces. Consequently, *Mockingbird* requires that the full traffic burst sequence be known before generating an associated adversarial example. This requirement is also in the Walkie-Talkie defense [15], along with several more expensive defenses. It means that the defense must maintain a database of relatively recent reference burst sequences for sites of interest. The responsibility for gathering and maintaining this database would most appropriately be given to the Tor network. In particular, special directory servers could be deployed to maintain the database and distribute fresh reference sequences periodically to clients. The servers could also be used to distribute pre-trained detector models to clients. The clients can then use this data to generate adversarial traces locally before each site visit.

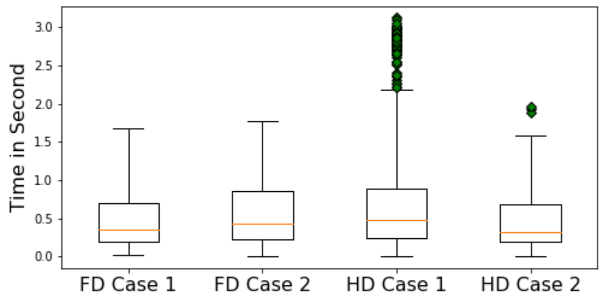



Fig. 12. Time to generate adversarial traces in *Mockingbird*.

Research on *adversarial patches* shows that adversarial examples can be generated that are agnostic to a particular target distribution and yet cause the source class to be misclassified by the model [45], [46]. We leave investigating applying this idea to network traffic traces as potential future work.

*2) Padding Mechanism:* Padding mechanisms must be designed so they can manipulate a trace sequence to match an adversarial sequence. To address this problem, *burst molding* must be performed by both the client and bridge, similar to that of the W-T defense [15]. Burst molding is difficult to achieve in a live setting, as padding must be added to the ends of bursts, which are difficult to reliably identify. To address this, we propose that a burst molding mechanism hold packets in a queue for each burst. The burst of real packets is forwarded normally until no additional packets are received for transmission. After a timeout is triggered, the queue is dumped onto the network. The size of the next burst can be easily communicated from the client to the bridge by embedding this information in the dummy packets of the prior burst. In this way, bursts can be easily molded to their appropriate size at the cost of a small additional delay. We leave the engineering challenge of setting the timeout for minimal added delay to future work.

*3) Computational Requirements:* The adversarial trace generation process currently requires hundreds of iterations of perturbing the reference trace and checking the resulting trace against the detector. For *Mockingbird* to be deployable, the full generation process must take at most a few seconds before each visit. We can see the generation time of FD and HD data by *Mockingbird* from Figure 12. For most of the sites, *Mockingbird* can generate the trace in around 0.5 second. Some sites in HD data, however, take up to 3 seconds.

As previously noted, significant computational resources (e.g. a dedicated GPU device) are currently necessary to generate the trace within this time limitation. Fortunately, there are possible solutions to this problem.

First, to save computation on the first visit to a site, we could relax the bandwidth constraint and enable the client to select a significantly different traffic pattern from the original with a single step. For sites that the user has visited previously, it would be possible to compute additional trace patterns in the background and save them for future use.

Second, works in other domains have explored techniques that allow models to be run efficiently by low-resource entities such as mobile devices. Techniques such as model pruning [47], [48] and data quantization [49], [50] have been shown to provide significant speedups in resource-constrained environments without requiring extensive changes to the underlying model. In particular, Yu *et al.* [48] demonstrate network quantization can achieve greater than 100% speedup with only 2% accuracy degredation on a MobileNet classifier. Similarly, Jacob *et al.* [50] achieved up to a 67% reduction in model computations using their parameter pruning technique on an AlexNet classifier. Additionally, techniques such as knowledge distillation [51] can be used to train new, more efficient student models with minimal reductions in accuracy. Application of these techniques may achieve speedups in the range of 100-300% for our detector model.

Finally, recent research by Chen *et al.* [52] proposes a *Sub-LInear Deep learning Engine* (SLIDE) which enable a deep-learning model to run in a CPU even faster than a GPU, indicating that it is possible to deploy *Mockingbird* in a realistic setting without requiring a dedicated GPU.

We leave further investigation of these techniques to future work.

### C. Server-Side Defense

Cherubin *et al.* [27] proposed to apply website fingerprinting defenses at the application layer, which is especially beneficial to onion services that are accessed only through Tor. A server-side application of *Mockingbird* could be an effective and practical way to defend a site from attacks. It would involve the website operator running a tool to examine their site trace from Tor, running *Mockingbird* to generate a set of effective traces, and then adding dummy web objects and padding existing web objects to specific sizes (following the techniques of Cherubin *et al.* [27]) to modify the network trace as needed. A web server can generate traces as needed during down times. Also, depending on the threat model of the site, it may only need to create a new trace periodically, such as once per day or per hour. At that rate of change, the attacker would need to download a new set of traces very frequently, increasing the attack cost and potentially making the attack less stealthy.

## VIII. Conclusion

We propose *Mockingbird*, a WF defense that offers better protection and lower bandwidth overhead than WTF-PAD and Walkie-Talkie, the previous state-of-the-art lightweight defenses. *Mockingbird* uses a novel mechanism to create adversarial traces that are robust even against adversarial training. It drops the Top-1 accuracy of the best attack from over 98% to at most 62% and cuts the Top-2 accuracy to at most 72%, which is much better than the Top-2 accuracies against WTF-PAD and Walkie-Talkie (95% and 97%, respectively). Furthermore, *Mockingbird*'s full-duplex bandwidth overhead is 58%, which is lower than either of the prior defenses. The results of information leakage analysis are in line with our previous conclusions when evaluating raw accuracy metrics. We emphasize that our experiments are conducted in the closed-world setting, where the attacker knows that the Tor client is assumed to visit one of the monitored sites. In a more realistic *open-world* setting, where the client could visit any site on the Web, 58% accuracy is very likely to lead to many false positives for the attacker.

Although *Mockingbird* has implementation challenges that must be addressed before it could be practically deployed in Tor, it shows the significant potential of an approach inspired by adversarial examples. Furthermore, it may be possible to leverage *Mockingbird* for server-side defense in the near future.

*Resources:* The code and datasets of this paper are available at: https://github.com/msrocean/mockingbird/.

## ACKNOWLEDGMENT

The authors would like to thank all the anonymous reviewers for their valuable reviews and feedback in this work.

## REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2012, pp. 1097–1105.

[2] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[3] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The kitti vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2012, pp. 3354–3361.

[4] C. Szegedy *et al.*, "Intriguing properties of neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2013.

[5] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 125–136.

[6] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, and M. Wright, "Tik-Tok: The utility of packet timing in website fingerprinting attacks," *Privacy Enhancing Technol. Symp. (PETS)*, vol. 2020, no. 3, pp. 5–24, 2020.

[7] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet fingerprinting: More practical and portable Website fingerprinting with N-shot learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2019, pp. 1131–1148.

[8] S. Bhat, D. Lu, A. Kwon, and S. Devadas, "Var-CNN: A data-efficient website fingerprinting attack based on deep learning," *Privacy Enhancing Technol. Symp. (PETS)*, vol. 2019, no. 4, pp. 292–310, 2019.

[9] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2018, pp. 1928–1943.

[10] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[11] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," in *Proc. 13th Workshop Privacy Electron. Soc. (WPES)*, 2014, pp. 131–134.

[12] X. Cai, R. Nithyanand, and R. Johnson, "CS-buFLO: A congestion sensitive website fingerprinting defense," in *Proc. 13th Workshop Privacy Electron. Soc. (WPES)*, 2014, pp. 121–130.

[13] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2014, pp. 227–238.

[14] M. Juárez, M. Imani, M. Perry, C. Díaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *Proc. Eur. Symp. Res. Comput. Secur. (ESORICS)*, 2016, pp. 27–46.

[15] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive Website fingerprinting attacks," in *Proc. USENIX Secur. Symp.*, 2017, pp. 1375–1390.

[16] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 39–57.

[17] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 506–519.

[18] A. Kurakin, D. Boneh, F. Tramèr, I. Goodfellow, N. Papernot, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[19] M. Imani, M. S. Rahman, and M. Wright, "Adversarial traces for website fingerprinting defense," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 2225–2227.

[20] S. Li, H. Guo, and N. Hopper, "Measuring information leakage in website fingerprinting attacks and defenses," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 1977–1992.

[21] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for Website fingerprinting," in *Proc. USENIX Secur. Symp.*, 2014, pp. 143–157.

[22] A. Panchenko *et al.*, "Website fingerprinting at Internet scale," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 1–15.

[23] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable Website fingerprinting technique," in *Proc. USENIX Secur. Symp.*, 2016, pp. 1187–1203.

[24] K. Abe and S. Goto, "Fingerprinting attack on Tor anonymity using deep learning," in *Proc. Asia–Pacific Adv. Netw. (APAN)*, vol. 42, 2016, pp. 15–20.

[25] 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 *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 332–346.

[26] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *Proc. Eur. Symp. Res. Comput. Secur. (ESORICS)*, 2006, pp. 18–33.

[27] G. Cherubin, J. Hayes, and M. Juarez, "Website fingerprinting defenses at the application layer," *Privacy Enhancing Technol. Symp. (PETS)*, vol. 2017, no. 2, pp. 186–203, 2017.

[28] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 274–283.

[29] N. Carlini and D. A. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *Proc. 10th ACM Workshop Artif. Intell. Secur. (AISec)*, 2017, pp. 3–14.

[30] N. Carlini and D. Wagner, "MagNet and 'Efficient defenses against adversarial attacks' are not robust to adversarial examples," 2017, *arXiv:1711.08478*. [Online]. Available: https://arxiv.org/abs/1711.08478

[31] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "A general framework for adversarial examples with objectives," *ACM Trans. Privacy Secur.*, vol. 22, no. 3, pp. 1–30, Jul. 2019.

[32] K. Eykholt *et al.*, "Robust physical-world attacks on deep learning visual classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1625–1634.

[33] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," 2016, *arXiv:1611.02770*. [Online]. Available: http://arxiv.org/abs/1611.02770

[34] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[35] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2014.

[36] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[37] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 372–387.

[38] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 582–597.

[39] (2017). *Alexa*. [Online]. Available: http://www.alexa.com

[40] M. K. Wright, M. Adler, B. N. Levine, and C. Shields, "The predecessor attack: An analysis of a threat to anonymous communications systems," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 4, pp. 489–522, 2004.

[41] M. K. Wright, M. Adler, B. N. Levine, and C. Shields, "Passive-logging attacks against anonymous communications systems," *ACM Trans. Inf. Syst. Secur.*, vol. 11, no. 2, pp. 1–34, Mar. 2008.

[42] G. Danezis, "Statistical disclosure attacks," in *Proc. IFIP Int. Inf. Secur. Conf.* Boston, MA, USA: Springer, 2003, pp. 421–426.

[43] N. Mallesh and M. Wright, "An analysis of the statistical disclosure attack and receiver-bound cover," *Comput. Secur.*, vol. 30, no. 8, pp. 597–612, Nov. 2011.

[44] G. Cherubin, "Bayes, not Naïve: Security bounds on Website fingerprinting defenses," *Privacy Enhancing Technol. Symp. (PETS)*, vol. 2017, no. 4, pp. 215–231, 2017.

[45] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1765–1773.

[46] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," 2017, *arXiv:1712.09665*. [Online]. Available: http://arxiv.org/abs/1712.09665

[47] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization, and Huffman encoding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[48] R. Yu *et al.*, "NISP: Pruning networks using neuron importance score propagation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9194–9203.

[49] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[50] B. Jacob *et al.*, "Quantization and training of neural networks for efficient Integer-Arithmetic-Only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[51] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," 2016, *arXiv:1612.03928*. [Online]. Available: http://arxiv.org/abs/1612.03928

[52] B. Chen, T. Medini, J. Farwell, S. Gobriel, C. Tai, and A. Shrivastava, "SLIDE: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems," 2019. *arXiv:1903.03129*. [Online]. Available: https://arxiv.org/abs/1903.03129

**Mohammad Saidur Rahman** (Graduate Student Member, IEEE) received the bachelor's degree from the Department of Management Information Systems, University of Dhaka, Bangladesh, and the M.S. degree from the Department of Computing Security, Rochester Institute of Technology (RIT), where he is currently pursuing the Ph.D. degree in computing and information sciences. He is also affiliated with the Department of Computing Security and the Global Cybersecurity Institute (GCI), RIT. His research interests include security and privacy, deep learning, lifelong learning, and adversarial machine learning.

**Mohsen Imani** (Member, IEEE) received the B.S. degree in electrical engineering from the Shahrood University of Technology, Iran, the M.S. degree in electrical engineering from the Iran University of Science and Technology, Iran, and the Ph.D. degree in computer science from The University of Texas at Arlington. He is currently a Principal Data Scientist at Anomali Inc. His research interests include security and privacy, machine learning in security and privacy, privacy enhancing technologies, and anonymous communications.

**Nate Mathews** (Associate Member, IEEE) received the B.S. degree in computing security from the Rochester Institute of Technology (RIT) in 2019, where he is currently pursuing the Ph.D. degree with the Golisano College of Computing and Information Sciences. He is also affiliated with the Department of Computing Security and the Global Cybersecurity Institute (GCI), RIT. His research interests include adversarial machine learning and privacy.

**Matthew Wright** (Senior Member, IEEE) received the B.S. degree in computer science from the Harvey Mudd College, and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Massachusetts, in 2002 and 2005, respectively. His dissertation work examined attacks and defenses of systems that provide anonymity online. He is currently the Director of Research at the Global Cybersecurity Institute (GCI), Rochester Institute of Technology (RIT), where he is also a Professor of Computing Security. His other interests include adversarial machine learning and understanding the human element of security. He has been the lead investigator on over $3.7 million in funded projects, including the NSF CAREER Award, and he has published 100 peer-reviewed papers, including numerous contributions in the most prestigious venues focused on computer security and privacy.