Swallow: A Transfer-Robust Website Fingerprinting Attack via Consistent Feature Learning

Meng Shen
Beijing Institute of Technology
Beijing, China
shenmeng@bit.edu.cn

Jinhe Wu
Beijing Institute of Technology
Beijing, China
jinhewu@bit.edu.cn

Junyu Ai
Beijing Institute of Technology
Beijing, China
aijunyu@bit.edu.cn

Qi Li
Tsinghua University
Beijing, China
qli01@tsinghua.edu.cn

Chenchen Ren
Shandong University
Jinan, China
chenchenren@mail.sdu.edu.cn

Ke Xu
Tsinghua University
Beijing, China
xuke@tsinghua.edu.cn

Liehuang Zhu
Beijing Institute of Technology
Beijing, China
liehuangz@bit.edu.cn

Abstract

Website fingerprinting (WF) attacks on Tor networks can analyze traffic patterns to identify the websites Tor users are visiting, and thus pose a significant threat to user privacy. In a real-world environment, Tor users face diverse network conditions and can also employ WF defenses, raising new challenges to launch WF attacks. The state-of-the-art (SOTA) WF attacks either rely on a strong assumption that WF classifiers are trained and deployed under the same network condition, or suffer from significant performance degradation against WF defenses. In this paper, we propose *Swallow*, a transfer-robust WF attack that can quickly transfer to new network conditions while maintaining robustness against various WF defenses. Specifically, we propose a novel trace representation named Consistent Interaction Feature (CIF), which aligns traffic distributions across different network conditions to capture consistent features. Then we design three data augmentation algorithms to simulate potential variations under various network conditions. We extensively evaluate Swallow using ten datasets, including both self-collected and public datasets. The closed- and open-world evaluation results demonstrate that Swallow significantly outperforms the SOTA attacks. In particular, with only 5 labeled instances per website for model fine-tuning, Swallow achieves an average improvement in accuracy of 17.50% over the SOTA WF attacks.

CCS Concepts

• **Networks** → **Network privacy and anonymity**.

Keywords

Tor; privacy; website fingerprinting; data augmentation

ACM Reference Format:
Meng Shen, Jinhe Wu, Junyu Ai, Qi Li, Chenchen Ren, Ke Xu, and Liehuang Zhu. 2025. Swallow: A Transfer-Robust Website Fingerprinting Attack via Consistent Feature Learning. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3744795

CCS '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3744795

1 Introduction

As an anonymous proxy tool, Tor has been increasingly adopted by users who seek to bypass censorship and safeguard their privacy [7]. Active daily users of Tor networks have exceeded 2 million by November 2024 [24]. Tor encrypts user traffic and routes it through a Tor circuit consisting of three relay nodes, effectively concealing the original source and making online activities difficult to trace. However, Tor has been shown to be vulnerable to various Website Fingerprinting (WF) attacks [26, 27, 31], where the attackers extract Tor traffic features (e.g., packet direction, length and timestamp) and leverage machine learning techniques to train a classifier, allowing accurate identification of the websites users are visiting.

Launching WF attacks in real-world scenarios remains a highly challenging task. The network conditions of the users are often assumed to be fixed [3, 15, 26, 27, 31], and the WF classifiers are trained and tested using datasets collected under the same conditions. In real world environment, however, Tor users face diverse network conditions (e.g., locations of guard relays, web browsers), which can lead to significant changes in Tor traffic patterns [36]. For instance, Tor users can connect through guard relays located in different regions, resulting in substantial variations in communication latencies [24]. When training and testing traces are collected under different network conditions, the accuracy of WF attacks drops significantly [15]. To adapt to new network conditions, attackers often need to collect a large number of labeled traffic instances for retraining, which is quite time consuming [2, 32]. Tor users can

also employ different WF defenses, such as WTF-PAD [16] and Front [8], to protect their traffic against WF attacks. These defenses can modify the original Tor traffic patterns by introducing dummy packets or delays, reducing the accuracy of WF classifiers [8, 16, 28].

Researchers have proposed various WF attacks that can be roughly classified into two categories: traditional attacks [3, 26, 27, 31] and transferable attacks [2, 32]. Traditional attacks assume that the training and testing conditions are the same. When applying under a different condition, they require a time-consuming data collection process for model retraining. Transferable attacks TF [32] and NetCLR [2] attempt to make the classifier transferable to new network conditions. They represent a Tor trace by a sequence of packet directions (e.g., +1 and −1 for incoming and outgoing packets, respectively). Since this representation is not robust against WF defenses [28], it suffers from a significant performance degradation against WF defenses. For example, their accuracy drops by more than 50% against Front (see Section 6.2). Therefore, practical WF attacks should be *transfer-robust*, meaning that they can achieve higher accuracy than existing WF attacks on defended traces across different network conditions.

In this paper, we propose *Swallow*[1], a transfer-robust WF attack that can quickly transfer to new network conditions while demonstrating superior robustness against WF defenses compared to SOTA WF attacks. We observe that the packet distributions of defended traces vary under different network conditions. The basic idea of Swallow is to align the packet distributions under different network conditions to learn consistent features.

Swallow is composed of three key modules. *First*, we propose a consistent feature representation module to convert raw website traces into Consistent Interaction Feature (CIF), which aligns traffic distributions across different network conditions to capture consistent features. *Second*, we develop a robust data augmentation module to simulate unseen traffic distributions under various network conditions. Since it is difficult to enumerate all kinds of changes in network conditions, we resort to website loading time as an indicator to reflect variation in traffic distribution, which can be roughly classified in three scenarios: stable, shorter, and longer loading time. For each scenario, we design the corresponding data augmentation strategy, namely trace fluctuation, trace aggregation and trace flatten. *Finally*, we design a few-shot website identification module to adapt the model to new network conditions using only a few labeled traffic instances. Specifically, we use self-supervised learning to train the encoder to capture consistent features by minimizing the distance between the original trace representation and the corresponding augmented trace representations in the feature space. This training process does not rely on labeled traffic instances. The trained encoder can then be fine-tuned for a new network condition using only a few labeled instances.

We collect eight datasets under different conditions and conduct extensive experiments with these datasets and two public datasets, i.e., Wang100 [35] and DF95 [31]. We compare the performance of six SOTA WF attacks i.e., DF [31], Tik-Tok [26], Var-CNN [3], RF [27], TF [32] and NetCLR [2] against seven different WF defenses i.e., WTF-PAD [16], Front [8], Surakav [9], RegulaTor [13] , Palette [28], Tamaraw [4] and TrafficSilver [6]. The results show that Swallow achieves the best attack performance in all scenarios. In particular, when fine-tuned with 5 labeled instances per website, Swallow achieves an average improvement in accuracy of 17.50% over the SOTA WF attacks.

We summarize our contributions as follows:

- We propose *Swallow*, a transfer-robust WF attack that can quickly transfer to new network conditions while demonstrating superior robustness against WF defenses compared to SOTA WF attacks. Specifically, we propose a novel trace representation named Consistent Interaction Feature (CIF) and design three data augmentation algorithms that are tailored to variation in traffic distributions.
- We collect eight datasets under different conditions (e.g., locations of guard relays, web browsers, and collection times), including seven closed-world datasets and one open-world dataset. The closed-world datasets consist of 100 websites, while the open-world dataset comprises 4,000 websites.
- We evaluate Swallow in both closed-world and open-world scenarios using our collected datasets as well as public datasets. The results demonstrate that Swallow significantly outperforms existing WF attacks in all scenarios. We release the dataset and source code of Swallow[2].

The remainder of the paper is organized as follows. We first introduce the related work in Section 2 and the threat model in Section 3. We present high-level design of Swallow in Section 4 and the design details in Section 5. Next, we conduct a comprehensive evaluation on the performance of Swallow in Section 6. We discuss relevant issues in Section 7 and conclude this paper in Section 8.

2 Related Work

2.1 WF Attacks

Existing WF attacks can be roughly classified into two categories: traditional attacks and transferable attacks.

Traditional WF attacks. Recent studies [3, 26, 27, 29, 31] use a simple representation of raw traffic traces (e.g., packet direction sequence) as input and then leverage deep neural networks (DNNs) to automatically extract traffic characteristics to avoid sophisticated feature engineering in WF attacks. These attacks are based on traditional supervised learning, assuming that the training and test datasets are collected under the same network conditions. However, when attacks are launched under a new network condition, they usually require time-consuming collection of a large number of well-labeled traffic traces to retrain the classifier.

Here, we review four typical attacks in this category. DF [31] uses the packet direction sequence as traffic representation and leverages Convolutional Neural Networks (CNNs) to build a classifier. Tik-Tok [26] further improves the attack accuracy of DF by using a different traffic representation that combines packet timing and direction information. Var-CNN [3] uses a ResNet18 model to process packet direction and timing information separately and trains an ensemble of website fingerprinting classifiers to enhance attack performance. RF [27] proposes a trace representation named

[1]Swallows are migratory birds known for their ability to travel long distances and quickly adapt to new environments with agility and efficiency.

[2]https://github.com/wujinhe0814/Swallow

Traffic Aggregation Matrix (TAM) that to improve robustness of WF attacks against existing defenses.

Transferable attacks. These attacks employ metric learning [32] or contrastive learning [11, 17] to fine-tune a well-trained classifier with a few traffic traces collected in a new network condition. During the pre-training phase, they aim to create a feature space where traffic traces from the same website are closely grouped, while those from different websites are clearly separated. In the fine-tuning phase, they further generalize to diverse network conditions by refining the feature space with only a few labeled traffic instances.

Here, we review two representative attacks in this category. TF [32] is composed of two parts, i.e., the feature extraction network based on CNN and k-NN classifier. It can achieve transferring tasks from one dataset to another using a few instances. NetCLR [2] is a contrastive learning-based WF attack that uses packet direction sequences as trace representation. It can adapt to new network conditions with only a few labeled traffic instances.

2.2 WF Defenses

To defend against WF attacks, recent studies have proposed various WF defenses, which can be roughly divided into three categories: *obfuscation*, *regularization* and *splitting* [28].

The basic idea of obfuscation defense is to randomize the sending of packets, ensuring that the traffic patterns for each visit to the same website are as different as possible. This category includes defenses such as WTF-PAD [16] and Front [8]. Regularization defense aims to regulate the traffic of all websites in a predefined pattern, ensuring that the traffic patterns of different websites are as similar as possible. This category includes four typical defenses: Tamaraw [4], Surakav [9], RegulaTor [13], and Palette [28]. Splitting defense is designed to split traffic and destroy the original features of websites without introducing time or bandwidth overhead. TrafficSliver is a representative splitting defense, which splits traffic over several Tor circuits in a highly random manner and merges it to the guard. It employs two strategies: BD and BWR. BD uses separate circuits for the incoming and outgoing packets, while BWR, the optimal strategy [6], weighs the selection of guard nodes to send Tor packets.

3 Threat Model

The threat model for transfer-robust WF attacks is illustrated in Figure 1. Tor relay nodes are geographically distributed throughout the world. To visit websites anonymously, a Tor client picks a Guard Relay and constructs a Tor circuit consisting of three Relay nodes to deliver traffic to the corresponding web servers. WF attack is usually considered a traffic classification problem. Previous studies [3, 26, 27, 31] assume that a WF attacker can train a classifier under the same conditions as the victim. Unlike them, we consider a more realistic scenario, where the WF attacker aims to apply a classifier trained under a certain network condition on a victim under a different network condition. Similar to previous WF attacks [3, 26, 27, 31], we consider a local and passive attacker, who can only collect traffic exchanged between victim and Tor guard relays but lacks the capability to modify or decrypt packets. The adversary may be the administrator of the user's local networks, Internet Service Providers (ISPs), or Autonomous System (AS).

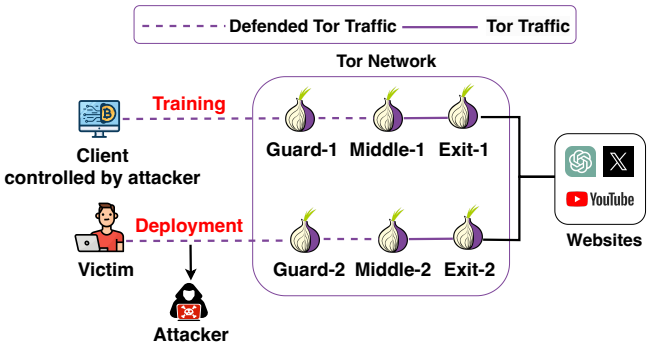


Figure 1: The threat model of transfer-robust WF attacks.

More specifically, the WF attacker extracts features from the collected traces of websites and trains a classifier offline. When launching WF attacks on victims with different conditions, the attacker collects a small number of traffic traces between the victim and its Guard Relay node in Tor network to fine-tune the trained classifier for identifying the website the victim is visiting. To mitigate WF attacks, the victim can deploy a WF defense to protect their connection privacy. Following prior work [2, 3, 26, 31, 32], we assume that the attacker has prior knowledge of the specific defense deployed by the user. Since all defenses are publicly available in Tor, it is challenging for users to deploy private defenses [28]. Thus, the attacker can collect traces generated under a specific defense and perform *adversarial training* to improve the accuracy of classifier against the corresponding defense.

Closed- and Open-World Scenarios. These scenarios are widely used to assess the effectiveness of WF attacks and defenses. In a closed-world scenario, the client is assumed to visit a limited set of websites referred to as *monitored* websites. The adversary has access to instances from this set to train a classifier aimed at distinguishing between these websites. In contrast, the open-world scenario, which is more representative of real-world conditions, involves the client visiting both monitored websites and a significantly larger number of *unmonitored* websites. The attacker, having access to only a subset of the unmonitored websites for training, attempts to determine if the client is visiting any monitored websites, and if so, identifies the specific website.

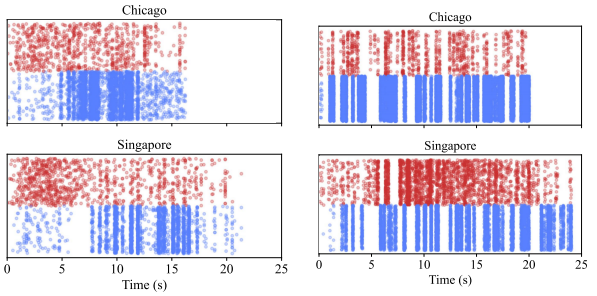
4 Design of Swallow

In this section, we present the key observation for our design and introduce the overview of Swallow.

4.1 Key Observation on Tor Traffic Distributions

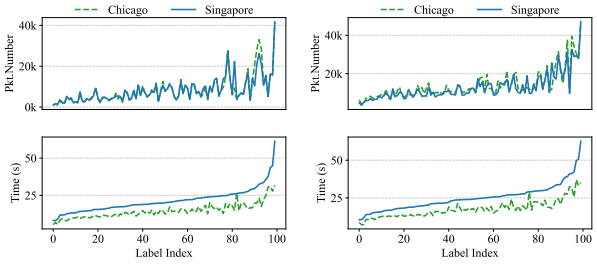
When a Tor user visits the websites through Tor networks, the resulting Tor traffic contains a sequence of outgoing and incoming packets. Tor traffic distribution roughly means the number of packets transmitted over time during the website loading process. Intuitively, the traffic distribution derived from visiting the same website may vary under different network conditions (e.g., via different Tor guard relays). WF defenses also have an impact on traffic distribution by introducing dummy packets or packet delay [8, 28].

To explore how the traffic distributions of the *defended traces* change under different network conditions, we collect undefended



(a) Defended Traces with Front (b) Defended Traces with Palette

Figure 2: Visualization of the distribution of defended traces derived from visiting the same website via two different guard relays located in Chicago and Singapore, respectively. For ease of illustration, the outgoing (•) and incoming (•) packets are plotted along Y-axis.



(a) Defended traces with Front (b) Defended traces with Palette

Figure 3: The average packet number and loading time of the website traces.

traffic under various network conditions and simulate the corresponding defended traces with a specific WF defense. Here, we select a representative obfuscation defense Front [8] and a representative regularization defense Palette [28]. We assume a client in Los Angeles and select guard relays located in different cities to visit the same set of websites (see Section 6.1 for more details on the collection of datasets). We randomly select defended traces that visit the same website from two guard relay locations (i.e., Chicago and Singapore) and visualize their distributions as shown in Figure 2. We observe that the website loading time through the guard relay located in Chicago is shorter than that in Singapore, which holds for defended traces with both Front and Palette.

To verify that a similar pattern exists in the defended traces for all 100 websites, we plot the average number of packets and the loading time for each website, as shown in Figure 3. In all subfigures, the website indices are ranked in ascending order according to their loading time measured in Singapore. We can observe that for the same WF defense, the average number of packets in the defended traces from two different locations is nearly the same, but the loading time shows significant differences. The loading time in Chicago is much shorter than in Singapore, which is consistent with the results in Figure 2. This is because the client is closer to Chicago, resulting in lower round-trip latency. We use Ping to measure the communication latency for LA-Chicago and LA-Singapore, which are 52.9 and 182.6 ms, respectively.

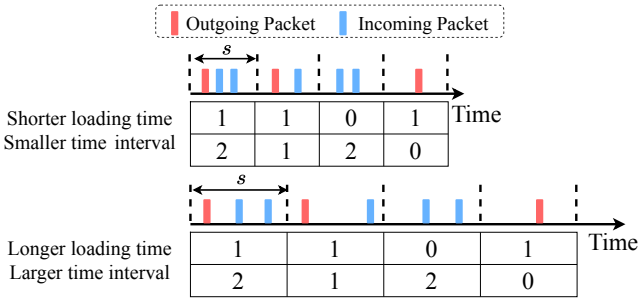


Figure 4: An illustration of traffic distribution alignment by flexibly adjusting the length of time interval s based on the loading time.

The above analysis indicates that the traffic distribution of the defended traces varies among different guard relays. Thus, the direction sequence of the packets used by the WF attacks TF [32] and NetCLR [2] experiences significant changes, making it difficult to achieve transferability when WF defenses are adopted. Intuitively, to achieve transferability for WF attacks, we can align traffic distributions to capture consistent traffic features that are robust under different network conditions. More specifically, if we count the number of outgoing and incoming packets transmitted in each time interval, we can flexibly adjust the length of time interval based on the loading time of traces to align different traffic distributions. As illustrated in Figure 4, the traffic distribution with a longer loading time is set with a larger time interval to align with the distribution with a shorter loading time. This motivates us to design a transfer-robust WF attack by learning consistent traffic features under different network conditions.

4.2 Overview of Swallow

In this subsection, we introduce Swallow. We design a novel trace representation called Consistent Interaction Feature (CIF) based on above observation. CIF aligns traffic distributions across different network conditions to capture consistent features. Then we propose data augmentation methods based on CIF to simulate traffic traces under various network conditions. Swallow leverages self-supervised learning to pre-train an encoder on unlabeled traffic instances, which can be fine-tuned with a few labeled samples to quickly adapt to new network conditions. Swallow consists of three modules, including Consistent Feature Representation, Robust Data augmentation, and Few-Shot Website Identification.

Consistent Feature Representation. The consistent feature representation module transforms raw traffic traces into a Consistent Interaction Feature (CIF). Based on the above observation, CIF divides the entire traffic traces into small time intervals, counting the number of outgoing and incoming packets per time interval. CIF dynamically adjusts the size of the time intervals based on the network condition of the website to align the Tor traffic distribution. The design details of this module are presented in Section 5.1.

Robust Data Augmentation. This module is designed based on our robust trace representation, CIF, which can effectively simulate the variations of defended traces under different network conditions. We design three data augmentation algorithms, each simulating a fundamental type of network scenario: minor network

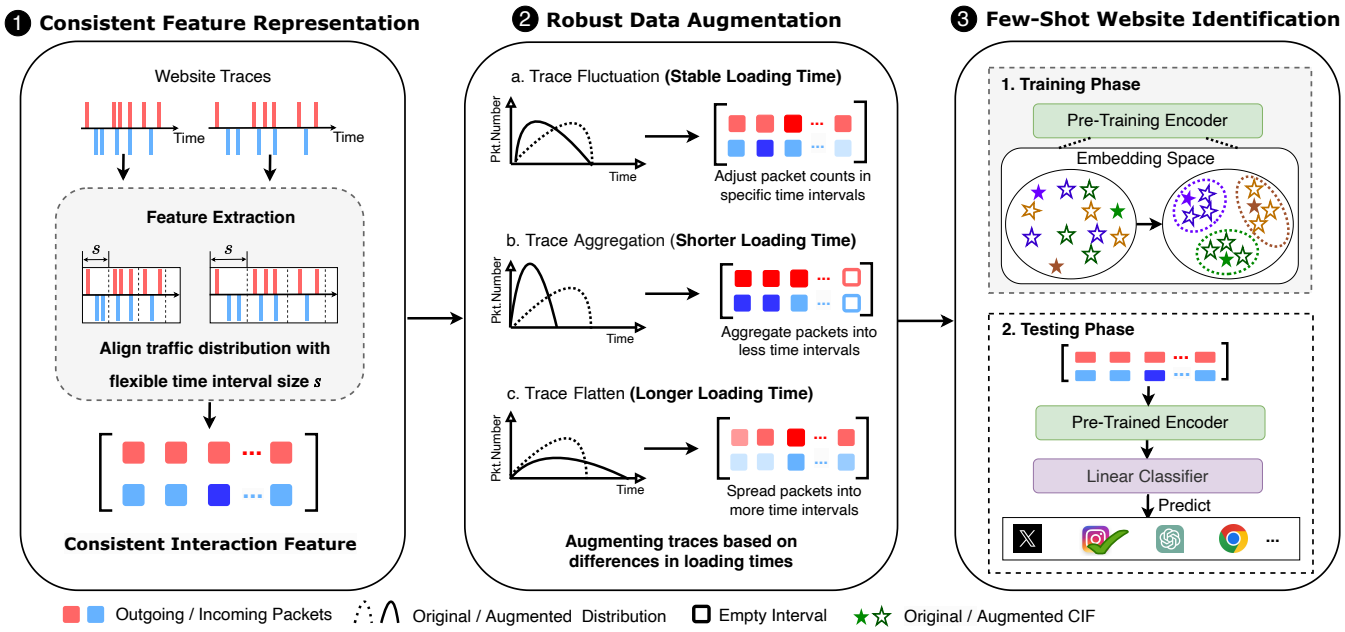


Figure 5: The overview of Swallow.

fluctuation, low-latency, and high-latency environments. These three scenarios correspond to three variations in website loading time: stable, shorter, and longer loading time. This module helps attackers improve the diversity of training data, thereby reducing the reliance on large amounts of labeled traffic instances. We will present the details in Section 5.2.

Few-Shot Website Identification. The few-shot website identification module leverages self-supervised learning to capture consistent features across diverse network conditions by ensuring the similarity between an original trace representation and its corresponding augmented trace representations. With these consistent features, the model achieves high attack accuracy in new network conditions, requiring only a few labeled traffic instances for fine-tuning. We will present the details in Section 5.3.

5 Design Details

In this section, we present the design details of Swallow, including the consistent feature representation module, the robust data augmentation module, and the few-shot website identification module.

5.1 Consistent Feature Representation

The consistent feature representation module transforms raw traffic traces into a CIF. We design this module based on a key observation that the packet distribution over time varies across network conditions, with shorter website loading times corresponding to denser packet distributions. For example, in low-latency network conditions, the website loading time is shorter, with higher peak packet counts in certain time intervals. This observation inspires us to divide the entire trace into smaller time intervals and count the number of outgoing and incoming packets in each interval. To align feature distributions across different network conditions, we propose CIF, which dynamically adjusts the time interval size based on the loading time of the traffic trace. Specifically, longer loading times, which typically indicate higher latency, trigger CIF to enlarge the time interval. This ensures that delayed packets are still captured within the current time interval.

We divide the entire trace into several small time intervals and dynamically adjust their sizes based on the website's loading time. To adapt to the model's input, the number of time intervals is fixed to N. The size of each time interval is defined as $s = \lceil \alpha \times \frac{T}{N} \rceil$. We introduce an adjustment factor α to ensure that the time interval size remains within a reasonable range. A reasonable time interval size is crucial because an excessively large interval reduces the amount of information that CIF can extract from the original traces, whereas an excessively small interval results in fewer packets within the window, making it more susceptible to network fluctuations. As shown in Figure 3, we perform a statistical analysis of the website loading times in our collected dataset. We observe that over 60% of the traces have a loading time of less than 20 seconds. We define both lower bound λ and upper bound μ bounds for the time interval size to prevent issues caused by excessively long or short website loading times.

The calculation process of CIF is shown in Algorithm 1. A visit to a certain website results in a traffic trace, which is denoted by $F = (f_1, f_2, \ldots, f_l)$, where l is the length of the trace. Let $f_k = \langle t_k, d_k \rangle$ be a tuple consisting of the packet timestamp and direction, where t_k and d_k are the arrival time and direction of the k-th packet, respectively. Note that d_k is 1 for an outgoing packet and -1 for an incoming packet. Let $M \in \mathbb{R}^{2\times N}$ denote the CIF of the trace F, where N is the number of time intervals considered in CIF. Assume that the size of each time interval is denoted by s, the loading time for a trace is T, and s can be calculated using $s = \lceil \alpha \times \frac{T}{N} \rceil$ (line 3). An element $m_{ij} \in M$ represents the number of incoming ($i = 1$) or outgoing ($i = 2$) packets whose timestamps are between $(j-1) \times s$

Algorithm 1: Calculation of CIF

Input: A traffic trace F, the trace loading time T, the number of columns of CIF N, the adjustment factor α, the lower / upper bound of time interval size λ,μ

Output: CIF $M = \{m_{ij} | i \in \{1, 2\}, j \in [1, N]\}$

1 Initialize the size of time interval s as 0;
2 Initialize the size of CIF M as $2 \times N$;
3 $s \leftarrow \alpha \times \frac{T}{N}$;
4 $s \leftarrow \max(\lambda, \min(s, \mu))$;
5 **for** *each packet* $f_k = \langle t_k, d_k \rangle \in F$ **do**
6 $\quad j \leftarrow \lfloor \frac{t_k}{s} \rfloor$;
7 $\quad$ **if** $j \leq N$ **then**
8 $\quad\quad i \leftarrow \begin{cases} 1 & \text{if } d_k < 0 \\ 2 & \text{otherwise} \end{cases}$;
9 $\quad\quad m_{ij} \leftarrow m_{ij} + 1$;
10 $\quad$ **end**
11 **end**
12 **return** M;

and $j \times s$ (line 6-9). The final returned M is considered as CIF. We evaluate the robustness of CIF in Section 6.4.

5.2 Robust Data Augmentation

The robust data augmentation module employs three augmentation algorithms to generate simulated traffic trace based on original traffic. Although CIF can effectively handle distribution differences in various network conditions, real-world traffic traces often contain small changes and anomalies that feature alignment may not fully capture. Data augmentation introduces additional variability, helping the model perform better when dealing with these subtle changes in the real world.

We propose robust data augmentation strategies to simulate variations in defended traces under different network conditions. These strategies are built upon CIF, which remains robust across diverse network conditions and defenses. Our approach addresses the limitations of existing methods. In computer vision, various data augmentation techniques [30], such as flipping, rotating, scaling, and cropping, have been widely adopted for images. However, these techniques are not directly applicable to traffic traces, as they fail to capture the unique variations introduced by different network conditions. The SOTA WF attack NetCLR [2] introduces NetAugment, which leverages packet direction sequences as the trace representation and converts them into bursts for data augmentation. The direction sequence is easily obfuscated by WF defenses [18], which makes NetAugment ineffective in simulating the variations of defended traces across different network conditions (see Section 6.4).

Specifically, we propose *RobustAugment*, which consists of three algorithms: *trace fluctuation*, *trace aggregation*, and *trace flatten*. Each algorithm simulates a fundamental type of network variation: minor network fluctuations, low-latency and high-latency environments. These three scenarios correspond to three variations in website loading time: stable, shorter, and longer loading time. The implementation details of the RobustAugment algorithm are shown in Algorithm 2. Following previous work [2], we do not modify

Algorithm 2: RobustAugment Algorithm

Input: Consistent Interaction Feature CIF, time interval size s, website loading time t

Output: Augmented Consistent Interaction Feature CIF_{aug}

1 Initialize an augment strategies list L = {Trace Fluctuation, Trace Aggregation, Trace Flatten};
2 Select a random strategy F from L;
3 $CIF_{aug} = F(CIF, s, t)$;
4 **return** CIF_{aug};

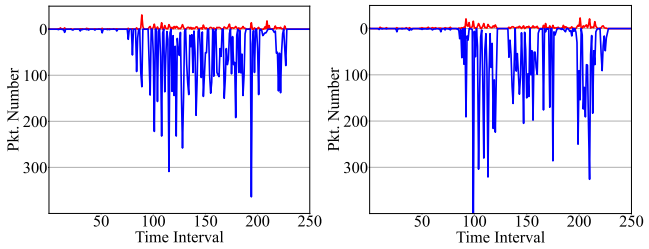


Figure 6: CIF patterns when browsing the same website under the same network condition twice.

the first 20 packets of each trace, as these packets are used for the initialization of the connection and the handshake, which remain consistent on different websites.

Trace Fluctuation. Even within the same network condition, multiple visits to the same website may exhibit different traffic patterns. This variation can result from minor network fluctuations during each visit or changes in the website's content. Figure 6 shows the CIF patterns when browsing the same website under the same network setting twice. It is evident that there are slight differences in the CIF traffic patterns between two visits. Specifically, one visit has a larger packet number in each time interval, while the other has a smaller packet number. To simulate this variation, we randomly modify the time interval value to generate augmented trace representation.

The trace fluctuation algorithm dynamically adjusts time interval values based on specific probabilistic parameters. If the time interval value is empty, the algorithm probabilistically replaces it with the average of neighboring intervals using a defined window size W_{modify} with probability $r_{padding}$. For non-zero time intervals, we randomly choose to either increase or decrease the time interval value. The modification process is controlled by the parameter r_{modify}. The implementation detail of the trace fluctuation algorithm is shown in Algorithms 3 in the Appendix.

Trace Aggregation. As mentioned in Section 4.1, the packet rate is faster in low-latency network conditions. Compared to high-latency conditions, the time interval value tends to be larger, and the website loading time is shorter. To simulate this variation, we aggregate packets into certain time intervals and remove others. Specifically, for each time interval, we remove it with a probability of r_{remove} to simulate shorter website loading times. We increase the time interval value with a probability of $r_{increase}$ for the time intervals that are retained. The implementation detail of the trace aggregation algorithm is shown in Algorithm 4 in the Appendix.

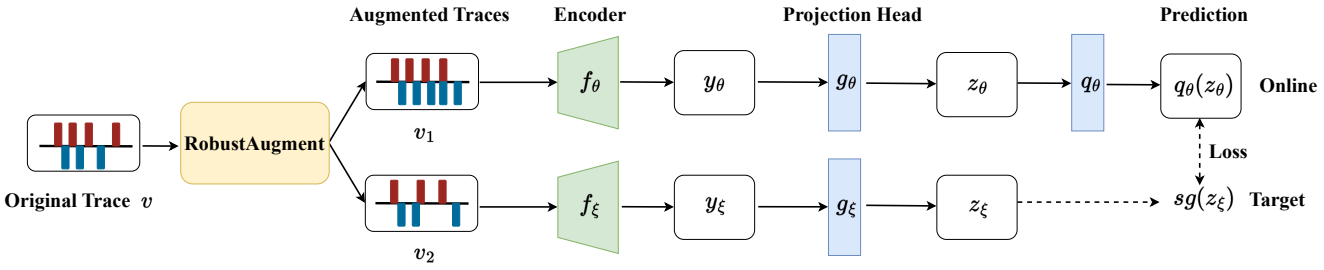


Figure 7: The pre-training phase of Swallow, which is based on a self-supervised learning framework BYOL [11].

Trace Flatten. In high-latency network conditions, the network becomes more congested, resulting in a lower packet transmission rate. This corresponds to a decrease in the time interval value, while the website loading time becomes longer. To simulate this variation, we flatten the entire trace and distribute packets across more time intervals. We use r_{insert} to control whether a time interval should be inserted. If so, a new time interval is inserted by averaging the surrounding values within a defined window W_{insert}. Additionally, we decrease the time interval value with a probability of $r_{decrease}$. The implementation detail of the trace flatten algorithm is shown in Algorithm 5 in the Appendix.

In summary, the above data augmentation simulates three scenarios that represent fundamental traffic variations: (1) minor fluctuations commonly observed in stable networks, (2) high-density traffic under low-latency conditions, and (3) dispersed traffic under high-latency conditions. They provide comprehensive coverage of typical network conditions.

5.3 Few-Shot Website Identification

To learn consistent features of network traffic and reduce reliance on a large number of labeled traffic samples, the few-shot website identification module leverages self-supervised learning to train an encoder that captures the similarity between the original trace representation and the augmented trace representation. Through this process, the encoder learns the consistent feature of traces under different network conditions, enabling attackers to achieve high attack accuracy in new network conditions with only a few labeled traffic instances for fine-tuning.

Swallow is based on a novel self-supervised learning framework BYOL [11], which reduces the distance between positive pairs and applies the skillfully-designed momentum mechanism to prevent the hidden space from collapsing without negative pairs. BYOL uses two neural networks to learn: the online and target networks. The online network is defined by a set of weights θ and is comprised of three stages: an encoder f_θ, a projector g_θ and a predictor q_θ. The target network has the same architecture as the online network but uses a different set of weights ξ. The target network supplies the regression targets necessary for training the online network. Its parameters, denoted by ξ, are maintained as an exponential moving average of the online network's parameters θ. Swallow consists of two stages: the pre-training phase and the fine-tuning phase.

Pre-Training Phase. The process of how the attackers train the pre-training encoder is shown in Figure 7. In the pre-training phase, we train an encoder to transform trace representation into low-dimensional embedding features. Specifically, two different augmented traces v_1 and v_2 are needed as the inputs of the online and target networks during the training phase. From the first augmented trace v_1, the online network outputs embedding features $y_\theta \triangleq f_\theta(v_1)$ and a projection head $z_\theta \triangleq g_\theta(y_\theta)$. The target network outputs $y_\xi \triangleq f_\xi(v_2)$ and the target projection head $z_\xi \triangleq g_\xi(y_\xi)$ from the second augmented trace v_2. We then output a prediction $q_\theta(z_\theta)$ of z_ξ and ℓ_2-normalize both z_θ and z_ξ to $\bar{z}_\theta \triangleq q_\theta(z_\theta)/\|q_\theta(z_\theta)\|_2$ and $\bar{z}_\xi \triangleq z_\xi/\|z_\xi\|_2$. Finally, we define the following mean squared error between the normalized predictions and target projections.

$$\mathcal{L}_{\theta,\xi} \triangleq \left\|\bar{q}_\theta(z_\theta) - \bar{z}_\xi\right\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \|z_\xi\|_2}. \quad (1)$$

We symmetrize the loss $\mathcal{L}_{\theta,\xi}$ in Eq. 1 by separately feeding v_2 to the online network and v_1 to the target network to compute $\tilde{\mathcal{L}}_{\theta,\xi}$. At each training step, we perform a stochastic optimization step to minimize $\mathcal{L}_{\theta,\xi}^{\text{Swallow}} = \mathcal{L}_{\theta,\xi} + \tilde{\mathcal{L}}_{\theta,\xi}$ with respect to θ only, but not ξ. After the training, we use the encoder f_θ to transform the trace representations into low-dimensional embedding features.

Fine-Tuning Phase. In the fine-tuning phase, the attackers use N labeled traffic traces to fine-tune the pre-trained encoder. We replace the projection head with a fully connected layer to output the probability for each class. Following prior work [2, 11], in the fine-tuning phase, we use the labeled dataset to fine-tune both the encoder and the fully connected layer.

Testing Phase. In the testing phase, the unknown traffic traces collected by the attacker are fed into the encoder to obtain low-dimensional embedding features. Then the trained fully connected layer predicts the labels of the unknown traffic traces.

6 Evaluation

In this section, we evaluate the performance of Swallow with public datasets and real-world datasets. We first describe the experimental settings in Section 6.1. Then we make a comprehensive comparison of Swallow with the SOTA attacks in both closed- and open-world scenarios in Sections 6.2 and 6.3, respectively. Finally, we conduct an ablation study on Swallow in Section 6.4.

6.1 Experimental Setup

Datasets. Considering that there are no public and comprehensive Tor datasets collected under different conditions (e.g., locations of guard relays, web browsers, and collection time), we construct

Table 1: Accuracy of WF attacks against defenses with 5 labeled traffic instances (Different Guard Relay Locations).

WF Attacks	Accuracy (%)								
	Undefended	WTF-PAD [16]	Front [8]	Surakav [4]	RegulaTor [13]	Palette [28]	Tamaraw [4]	UniDef[1]	TrafficSliver [6]
DF [31]	42.87	27.77	9.06	8.21	3.45	4.06	7.68	1.53	9.34
Tik-Tok [26]	56.61	51.35	16.23	20.76	3.38	3.45	5.74	2.09	12.43
Var-CNN [3]	49.40	13.77	4.57	20.68	1.49	2.23	1.57	1.47	4.21
RF [27]	69.21	54.93	47.57	14.04	29.72	5.27	3.91	3.62	35.72
TF [32]	86.23	73.97	28.99	29.54	1.88	2.24	8.20	1.32	15.18
NetCLR [2]	87.21	74.18	18.40	19.92	1.90	2.34	7.21	1.94	9.24
Swallow	**87.42**	**81.40**	**62.41**	**46.61**	**33.60**	**10.19**	**8.53**	**3.91**	**45.52**

[1] *UniDef* is a self-constructed defense where the distribution of Tor traffic for all websites is made uniform.

Table 2: Summary of Collected Datasets.

Dataset ID	Location	Browser	Time	Size
D_1	Chicago	TBB	2024-7	100 × 100
D_2	Singapore	TBB	2024-7	100 × 100
D_3	London	TBB	2024-7	100 × 100
D_4	Johannesburg	TBB	2024-7	100 × 100
D_5	Mumbai	TBB	2024-7	100 × 100
D_6	Chicago	TBB	2024-10	100 × 100
D_7	Chicago	Chrome	2024-7	100 × 100
D_8	Chicago	TBB	2024-7	4000 × 1

our own datasets using six cloud servers on Vultr [1], as shown in Table 2. Five of these servers are used as our private bridges (i.e., the guard relays), which are located in Chicago, Singapore, London, Johannesburg, and Mumbai. The rest server located in Los Angeles is used as the Tor client to visit websites via Tor browser (TBB version 10.5.10) or Chrome (version 112.0.5615.28).

We collect datasets in both closed- and open-world settings. Following previous studies [9, 28], the websites are selected from the Tranco [23] list, which combines five established rankings (i.e., Umbrella, Majestic, Farsight, CrUX, and Radar) while excluding undesirable domains, such as those that are unavailable or malicious. We select the first 100 websites on the list (i.e., the most popular websites) as the monitored sites, and each monitored website has at least 100 instances. We also randomly select 4,000 websites as unmonitored sites, where each site has only one instance.

In addition, we also use public datasets provided in recent studies to evaluate the performance of WF attacks.

Wang dataset [35]. The dataset includes both closed- and open-world websites. Closed-world websites are selected from lists of blocked sites in some countries, and open-world websites are chosen from the Alexa Top Sites. The dataset was collected in 2013 using TBB version 3.X.

- **Wang100**. The set of 100 closed-world websites, where each website has 90 traffic instances.
- **Wang9000**. The set of 9,000 open-world websites, where each website has 1 traffic instance.

DF dataset [31]. The dataset was collected in 2016 using TBB version 6.X, including websites in both closed- and open-world settings. All websites were chosen from the Alexa Top Sites.

- **DF95**. The set of 95 closed-world websites, where each website has 1,000 traffic instances.
- **DF40000**. The set of 40,000 open-world websites, where each website has 1 traffic instance.

Ethical Considerations. All guard relays used in our data collection are kept private and we do not accept requests from real Tor users. We limit the number of clients in parallel (8 in our setup) to reduce the burden on both the Tor network and the Web servers. Furthermore, we retain only the packet timestamps and directions, which are essential for our experiments, and we do not collect any additional information.

WF attacks. To make a comprehensive comparison, we select six SOTA WF attacks, namely DF [31], Tik-Tok [26], Var-CNN [3], RF [27], TF [32] and NetCLR [2]. They are all built with the source codes released by their authors. All WF attacks are trained and tested on a server equipped with an Intel Core i7 3.4 GHz, 32GB of memory, and a GeForce RTX 3080. To ensure a fair comparison, we perform parameter tuning of these attacks to achieve similar or better performance as reported in their original papers.

WF defenses. We select seven typical WF defenses, including WTF-PAD [16], Front [8], Surakav [9], RegulaTor [13], Palette [28], Tamaraw [4] and TrafficSliver [6]. We simulate each defense in the undefended dataset to create defended datasets for closed- and open-world evaluation.

6.2 Closed-World Evaluation

In this subsection, we evaluate the performance of attacks against WF defenses in *four* typical scenarios where training and testing of WF attacks are conducted under different conditions. Following the assumption of closed-world evaluation, the clients can only visit the set of monitored websites.

Scenario #1: Different Locations of Guard Relays. As mentioned in Section 4.1, the traffic patterns generated by visiting websites using different guard relays exhibit significant differences. For transferable WF attacks (e.g., TF [32], NetCLR [2] and Swallow), we pre-train the classifier on the *original* dataset D_1 (located in Chicago), and fine-tune the classifier with the *target* datasets D_2, D_3, D_4, or D_5, respectively. During fine-tuning, we randomly select N traffic instances (i.e., $N = \{5, 10, 15, 20\}$) of each website from the corresponding dataset, while the remaining instances are used for testing. For traditional WF attacks (e.g., DF [31], Tik-Tok [26], VarCNN [3], RF [27]), we directly train the classifier with the N traffic instances of each website from the target dataset D_2, D_3, D_4, or D_5 and test with the remaining instances, which is the same as in previous studies [2, 32]. To eliminate the impact of sample

Table 3: Accuracy of WF attacks against defenses with various labeled traffic instances (Concept Shift).

N	WF Attacks	Accuracy (%)						
		Undefended	WTF-PAD [16]	Front [8]	Surakav [4]	RegulaTor [13]	Palette [28]	TrafficSliver [6]
5	RF [27]	59.02	56.25	42.25	16.81	16.26	7.31	36.64
	TF [32]	84.98	79.94	10.96	31.45	3.01	2.77	13.97
	NetCLR [2]	86.51	79.44	9.03	22.43	3.28	2.74	9.02
	Swallow	**86.54**	**81.15**	**61.26**	**43.57**	**19.35**	**10.10**	**45.99**
10	RF [27]	72.62	72.31	62.18	37.51	21.09	10.46	48.51
	TF [32]	90.01	83.02	12.65	34.78	3.36	3.02	16.29
	NetCLR [2]	89.87	85.15	15.09	30.01	3.90	3.13	12.94
	Swallow	**90.06**	**86.62**	**76.48**	**52.97**	**23.54**	**13.08**	**56.55**
15	RF [27]	89.14	74.10	71.81	47.14	25.64	13.52	58.02
	TF [32]	90.48	83.64	12.82	36.59	3.64	3.57	17.28
	NetCLR [2]	92.00	87.66	19.39	35.34	3.50	3.49	17.20
	Swallow	**92.35**	**89.80**	**79.45**	**58.68**	**26.98**	**14.12**	**61.24**
20	RF [27]	89.75	83.97	75.35	55.78	27.15	16.10	61.58
	TF [32]	90.83	84.03	13.64	36.75	3.86	3.45	17.39
	NetCLR [2]	92.35	88.96	22.96	39.06	3.90	3.58	19.00
	Swallow	**93.49**	**91.15**	**83.69**	**60.84**	**27.94**	**16.23**	**66.47**

selection on performance, we repeat each experiment five times and obtain the average accuracy of each WF attack. In addition, we construct an ideal defense where the distribution of Tor traffic for all websites is made uniform, named *UniDef*. UniDef obfuscates the number of packets for all websites within each time interval to achieve a uniform distribution. However, it incurs over 200% bandwidth overhead and 40% time overhead.

The performance of WF attacks with only five traffic instances ($N = 5$) is shown in Table 1. Additional results for other sample sizes are provided in Table 10 within the Appendix. These results demonstrate trends and conclusions similar to those observed when $N = 5$. TF, NetCLR, and Swallow still achieve an accuracy greater than 85% on undefended traces with only five instances. This is because they have learned how to extract generalized features during the pre-training phase [11]. In contrast, traditional attacks require more labeled instances for training and thus achieve an accuracy below 70%.

For the obfuscation defenses WTF-PAD and Front, Swallow achieves the best accuracy, with accuracies of 81.40% and 62.41%, respectively. Particularly on Front, the accuracy of Swallow is higher than that of NetCLR, TF, RF, Var-CNN, Tik-Tok and DF by 44.01%, 33.42%, 14.84%, 57.84%, 46.18% and 53.35% , respectively. The performance of TF and NetCLR significantly declines on Front, where both achieve accuracies below 30%. This is because TF and NetCLR use direction sequence as their trace representation, which is not robust against Front (see Section 6.4).

The regularization defenses not only insert dummy packets but also introduce packet delay, providing stronger protection. Swallow still achieves the highest accuracy under this setting. On Surakav and RegulaTor, Swallow improves the baseline accuracy by an average of 30.13% and 28.76%, respectively, significantly outperforming other attacks. Both TF and NetCLR, the two SOTA transferable attacks, achieve accuracy of less than 3% on RegulaTor. Similarly, CUMUL, DF, Tik-Tok, and Var-CNN almost completely lose their classification ability against these defenses. Palette and Tamaraw provide even stronger protection, reducing the accuracy of almost all attacks to below 10%. Despite this, Swallow continues to achieve the best performance, with an average improvement in baseline accuracy of 7.14% and 2.85% on Palette and Tamaraw, respectively. UniDef reduces the accuracy of all WF attacks to below 5%, close to random guessing.

Splitting defenses, such as TrafficSliver, split website traffic to limit the traces obtained by attackers. Swallow achieves the best performance in this scenario, with an accuracy of more than 45% , Swallow improving the baseline accuracy by an average of 32.58%. Under this defense, TF and NetCLR achieve accuracy of 15.10% and 9.24%, respectively, which are over 30% lower than that of Swallow.

The results show that Swallow can efficiently transfer across different guard relays, achieving high attack accuracy with only 5 labeled traffic instances, with an average accuracy improvement of 20.23% compared to the SOTA WF attacks. This is attributed to CIF, which is more robust than other trace representations under different guard relays and defenses (see Section 6.4). The data augmentation further simulates network condition variations, enabling the encoder to learn consistent features.

Scenario #2: Concept drift. Tamaraw reduces the accuracy of all attacks to less than 10% by regularizing the packet sending. However, previous work [28] has shown it introduces high overheads. Similar to Tamaraw, UniDef also requires a significant number of dummy packets and delays, making real-world deployment impractical. Specifically, both defenses incur over 200% bandwidth overhead and 40% time overhead, burdening the Tor network and degrading user experience [20]. Therefore, they will not be considered in the subsequent experiments. Considering the poor performance of CUMUL, DF, TikTok, and Var-CNN in above scenario, these methods will not be included in the following experiments.

Concept drift means that changes in website content can lead to variations in traffic patterns over time. Previous works [2, 32] have demonstrated that concept drift reduces the accuracy of WF attacks. To evaluate the performance of WF attacks in dealing with concept drift, we conduct experiments with the collected datasets D_1 and D_6, where they were collected in July and October 2024, respectively. The website labels in both datasets are the same. Similar to the setting in Scenario #1, we pre-train the classifier on dataset D_1 and fine-tune it on dataset D_6. During the fine-tuning phase, we

Table 4: Accuracy of the SOTA WF attacks against defenses with various labeled traffic instances (Different Browsers).

N	WF Attacks	Accuracy (%)						
		Undefended	WTF-PAD [16]	Front [8]	Surakav [4]	RegulaTor [13]	Palette [28]	TrafficSliver [6]
5	RF [27]	24.68	11.74	10.43	3.19	2.11	5.36	4.87
	TF [32]	32.37	15.10	4.80	4.30	1.10	1.39	3.41
	NetCLR [2]	38.26	12.39	4.87	4.69	1.25	1.52	2.13
	Swallow	**69.09**	**32.17**	**25.84**	**15.06**	**6.39**	**7.35**	**8.39**
10	RF [27]	66.84	40.78	48.56	6.22	2.07	8.48	14.42
	TF [32]	36.50	15.21	5.88	5.18	1.14	1.39	3.33
	NetCLR [2]	50.86	19.61	8.20	5.61	1.26	1.79	2.81
	Swallow	**79.13**	**49.22**	**53.53**	**22.04**	**7.73**	**8.85**	**15.54**
15	RF [27]	72.78	50.14	57.76	13.34	4.45	8.52	18.88
	TF [32]	39.07	18.64	6.61	6.04	1.10	1.62	3.53
	NetCLR [2]	65.10	25.10	10.76	6.58	1.28	1.97	3.67
	Swallow	**81.00**	**52.48**	**60.52**	**25.05**	**8.93**	**9.17**	**21.32**
20	RF [27]	79.57	58.25	60.56	17.33	3.80	9.28	22.58
	TF [32]	40.56	17.73	6.84	6.22	1.18	1.66	3.90
	NetCLR [2]	65.47	28.01	12.91	7.17	1.28	2.00	4.14
	Swallow	**84.82**	**60.40**	**66.67**	**29.24**	**9.03**	**10.22**	**24.12**

Table 5: Summary of Different Data Distributions Datasets.

Dataset ID	Time	TBB Version	Size
Wang100	2013	3.X	100 × 90
Wang9000	2013	3.X	9000 × 1
DF95	2016	6.X	95 × 1000
DF40000	2016	6.X	40000 × 1

randomly select N traffic instances for each website and use the remaining instances for testing. The results are shown in Table 3.

Swallow outperforms the SOTA WF attacks in all settings. The attack accuracy of TF and NetCLR is comparable to that of Swallow on undefended traces, but decreases significantly under various defenses. An interesting observation is that RF demonstrates greater robustness than TF and NetCLR against different defenses, even though it does not consider transferability in its design. When more labeled instances are available ($N = 20$), its performance becomes comparable to that of Swallow. However, with only 5 labeled traffic instances, Swallow achieves an accuracy of 61.26% under Front, which is 19.01%, 50.30%, and 52.23% higher than that of RF, TF, and NetCLR, respectively. This demonstrates that Swallow outperforms RF when samples are limited, making it a more cost-effective solution for attackers to adapt to new network conditions.

Scenario #3: Different Browsers. When users visit websites through Tor, they generally use the Tor browser. However, some users also visit websites via the Chrome browser with the Tor plugin. There is a significant difference in traffic patterns when visiting the same website using these two methods [28]. The traffic volume of Chrome is much larger than that of the Tor Browser because the traffic generated by the Tor browser does not include features that potentially leak users' privacy (e.g., SPDY and HTTP/2) [21]. For transferable WF attacks (e.g., TF [32], NetCLR [2] and Swallow), we pre-train the classifier on dataset D_1 and fine-tune it on dataset D_7. For traditional WF attack RF [27], we directly train the classifier with the N traffic instances of each website from the dataset D_7 and test with the remaining instances, which is the same as in previous studies [2, 32]. The results are shown in Table 4.

Swallow still demonstrates its robustness, significantly outperforming the baseline across all settings. Specifically, with only 5 labeled undefened traffic instances, Swallow improves accuracy by 44.41%, 36.72%, and 30.83% compared to RF, TF, and NetCLR, respectively. Compared to the previous two scenarios, the attack performance of all WF attacks has decreased. This is because visiting websites using Tor Browser and Chrome results in significantly different feature distributions.

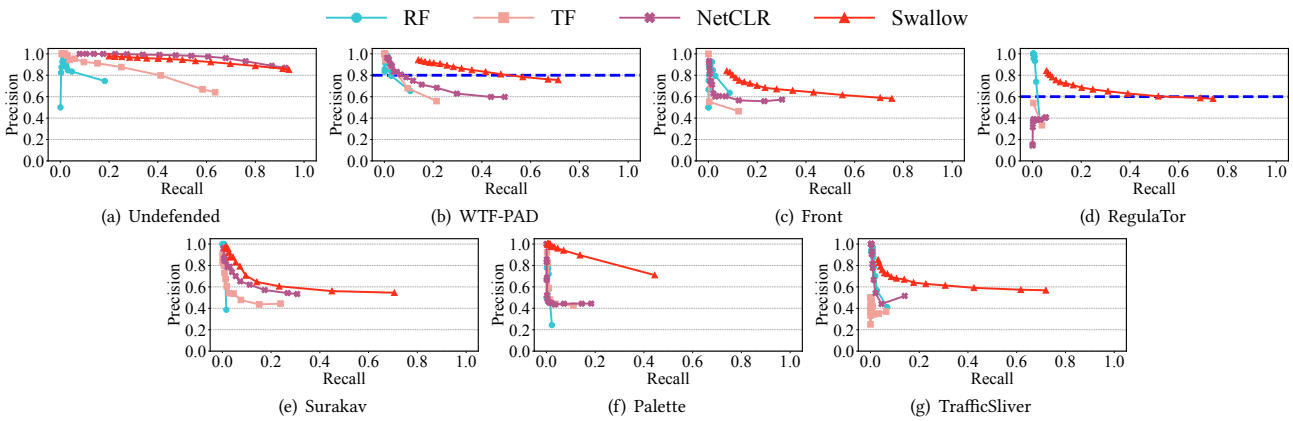
The accuracy of RF shows a significant decrease compared to other scenarios, even though both its training and testing are conducted on dataset D_7. This is because visiting the same website using the Chrome browser generates more packets compared to the Tor browser. These additional packets may mask the distinguishing features of the original website [28], reducing the differentiation between websites. Even with more labeled instances (N = 20), TF and NetCLR still achieve less than 20% accuracy on Front, Surakav, RegulaTor, Palette, and TrafficSliver. In contrast, Swallow achieves an average accuracy of 51.03% across these defenses.

Scenario #4 : Different Data Distributions. In the previous experiments, we assume that only one network condition differs while the website labels across all datasets remain consistent. To further evaluate the performance of the WF attacks in a more realistic scenario that covers cases in Scenario#1-#3, we conduct experiments on the publicly available datasets Wang100 and DF95. As shown in Table 5, there is a gap of 3 years in the collection time of these two datasets. Since the authors do not disclose the website labels in the dataset, we assume that the website labels are inconsistent. The two datasets utilize different Guard Relays to visit websites and also employ different versions of the TBB. These all ensure that the data distribution of these two datasets is significantly different. For transferable WF attacks (e.g., TF [32], NetCLR [32] and Swallow), we pre-train the classifier on dataset Wang100 and fine-tune it on dataset DF95. For traditional WF attack RF [27], we directly train the classifier with the N (i.e., $N = \{5, 10, 15, 20, 500\}$) traffic instances of each website from the dataset DF95 and test with the remaining instances, which is the same as in previous studies [2, 32].

As shown in Table 6, Swallow outperforms the SOTA WF attacks in few-shot settings, achieving the best attack performance when

Table 6: Accuracy of the SOTA WF attacks against defenses with various labeled traffic instances (Different Data Distributions).

N	WF Attacks	Accuracy (%)						
		Undefended	WTF-PAD [16]	Front [8]	Surakav [4]	RegulaTor [13]	Palette [28]	TrafficSliver [6]
5	RF [27]	54.37	44.72	35.29	20.65	14.69	8.95	26.04
	TF [32]	61.86	29.65	6.65	16.29	4.85	9.68	10.85
	NetCLR [2]	75.41	26.57	9.14	17.81	5.37	7.01	10.18
	Swallow	**75.91**	**64.25**	**37.80**	**46.77**	**15.94**	**11.72**	**38.28**
10	RF [27]	76.95	64.74	53.87	32.84	19.59	13.40	41.59
	TF [32]	65.57	34.22	7.68	17.64	5.27	10.33	12.19
	NetCLR [2]	84.44	40.26	12.80	24.36	5.66	8.97	13.29
	Swallow	**85.71**	**77.01**	**59.71**	**53.45**	**20.23**	**14.53**	**52.38**
15	RF [27]	85.11	75.59	64.92	42.78	23.22	14.53	50.34
	TF [32]	66.36	37.21	7.85	19.11	5.90	10.88	12.67
	NetCLR [2]	87.13	50.71	17.83	30.08	6.43	10.19	15.87
	Swallow	**87.27**	**76.04**	**68.03**	**55.93**	**24.73**	**14.70**	**55.14**
20	RF [27]	85.61	76.81	68.18	47.08	26.63	16.50	53.34
	TF [32]	67.21	38.03	8.26	20.04	5.93	11.50	13.15
	NetCLR [2]	89.92	56.76	21.07	33.18	6.72	11.10	17.51
	Swallow	**90.98**	**84.48**	**72.26**	**61.60**	**26.75**	**16.81**	**62.03**
500	RF [27]	98.14	96.01	93.06	78.36	52.68	33.72	67.11
	TF [32]	89.41	54.43	17.55	36.29	5.92	13.84	13.42
	NetCLR [2]	97.36	87.05	69.57	60.71	15.54	18.32	16.29
	Swallow	**97.55**	**91.13**	**90.96**	**71.55**	**49.56**	**31.36**	**64.38**



(a) Undefended (b) WTF-PAD (c) Front (d) RegulaTor

(e) Surakav (f) Palette (g) TrafficSliver

Figure 8: Precision-recall curves of WF attacks in the open-world scenario.

the number of samples is less than 20. Specifically, with only 5 labeled traffic instances, the accuracy of Swallow is higher than that of NetCLR, TF, RF by 19.88%, 21.55%, 12.28% on average. TF and NetCLR continue to show significant performance degradation against defenses. For instance, with five labeled traffic instances, their accuracies drop below 10% against Front, whereas Swallow achieves 37.80%. When there are sufficient labeled instance for training ($N = 500$), RF performs comparably to Swallow. This is because RF is based on supervised learning and requires a large number of labeled instances to train the classifier. However, its performance significantly declines under limited-instance conditions ($N = 5$). In contrast, the accuracy of the SOTA WT attack NetCLR on RegulaTor and Palette remains below 20% in this scenario.

6.3 Open-World Evaluation

In this subsection, we evaluate the performance of each attack in the open-world scenario. In the open-world scenario, users are free to visit any website. At the same time, there may be attackers who monitor a specific set of websites and use a trained classifier to determine whether the website visited by users falls within this monitored set. This is a more realistic and challenging scenario for attackers. Specially, the WF attackers use binary classification to identify monitored or unmonitored sites and then use the multi-class classification to recognize the specific monitored site. In this scenario, we use the same pre-trained model as in Scenario #4 of the closed-world setting. Then the model fine-tunes and tests on the closed-world dataset DF95 and the open-world dataset DF40000. Following previous work [2], we use a dataset of unmonitored websites that has an equal size to the monitored websites in the fine-tuning phase. We randomly select $N = 10$ instances from each website in the monitored set, means that there are 95×10 traffic instances selected from dataset DF95 and 950 traffic instances from DF40000. To evaluate the performance of the SOTA WF attacks, we select 95×50 monitored instances and 10,000 unmonitored instances in the testing phase.

Table 7: The MMD loss under different WF defenses.

	CUMUL [22]	Direction Sequence [2]	TAM [27]	CIF
WTF-PAD [16]	3.33	0.47	0.01	**0.01**
Front [8]	1.29	0.55	0.07	**0.04**
Surakav [9]	3.71	3.42	1.14	**0.86**
RegulaTor [13]	6.31	2.74	1.75	**1.63**
Palette [28]	3.43	0.91	0.83	**0.81**
Tamaraw [4]	5.67	4.12	2.50	**2.37**
TrafficSliver [6]	2.57	4.07	0.42	**0.40**
Average	3.76	2.32	0.96	**0.87**

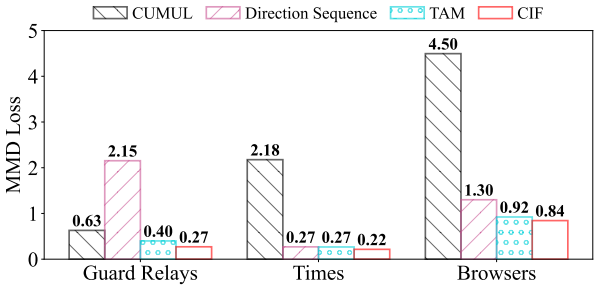


Figure 9: The MMD loss under different network conditions.

Given the significant imbalance between monitored and unmonitored sets, the precision-recall curve is frequently employed in the literature [27, 28, 31, 34]. If a monitored website is labeled correctly (i.e., the highest output probability exceeds a pre-defined threshold), it is classified as a True Positive (TP); otherwise, it is considered a False Negative (FN). Similarly, if an unmonitored website is incorrectly labeled as monitored, it is classified as a False Positive (FP); otherwise, it is a True Negative (TN). Precision and recall are then computed as TP / (TP + FP) and TP / (TP + FN), respectively. By adjusting the threshold, the attacker can control the trade-off between precision and recall.

The precision and recall curves of WF attacks tested on dataset DF95 and DF40000 are shown in Figure 8. Swallow still outperforms the SOTA WF attacks in this scenario. On undefended traces, Swallow achieves attack performance comparable to NetCLR. The attack performance of TF drops significantly when transferred to datasets with different feature distributions, making it less effective than Swallow and NetCLR. Similarly, RF performs worse in the open-world setting compared to the closed-world. This is due to its reliance on traditional supervised learning, which requires more labeled traffic instances to maintain effectiveness.

On defended traces, all attacks experience a significant performance drop except for Swallow. For instance, on the obfuscation-based defense WTF-PAD, when tuned for high precision (over 0.8), the recall of the other three attacks falls below 0.1, while Swallow maintains a recall greater than 0.4. RF, TF and NetCLR nearly lose their classification capability against regularization-based defense Surakav, RegulaTor and Palette and the splitting defense TrafficSliver. Their recall drops to almost 0 across different threshold settings, indicating that nearly all open-world websites are misclassified as closed-world websites. In contrast, Swallow still maintains a recall greater than 0.4 when precision is adjusted to 0.6.

The results show that Swallow can effectively distinguish defended traffic instances from monitored and unmonitored websites, even with limited labeled instances in the open-world scenario. Our carefully designed data augmentation algorithms enhance the model's ability to generalize to new network conditions, enabling it to effectively identify monitored websites under the interference of unmonitored websites.

6.4 Ablation Study

In this subsection, we use quantitative measures to evaluate the effectiveness of CIF and RobustAugment in transferring across different network conditions on defended traces. Then we also evaluate the impact of the three modules on the attack accuracy of the Swallow.

Effectiveness of CIF. We use Maximum Mean Discrepancy (MMD) loss [10] to measure the effectiveness of CIF under different WF defenses and network conditions that lead to variations in the traffic distribution of Tor. MMD loss is commonly used to measure the discrepancy between the feature distributions of the source and target domains [27]. A lower MMD loss indicates that the trace representation exhibits greater stability across different defenses and network conditions. We compare CIF with three typical trace representations used in SOTA WF attacks, namely CUMUL [22], Direction Sequence [2, 32], and TAM [27]. We simulate and generate traces for seven defenses on the D_1 dataset and calculate the MMD loss between the defended traces and the traces without defense.

As shown in Table 7, CIF achieves the lowest average MMD loss across all defenses, demonstrating that CIF is more robust compared to other trace representations. CUMUL and Direction Sequence exhibit significant instability when handling defenses, with average MMD loss of 3.76 and 2.32, respectively, while CIF maintains a significantly lower loss of 0.87. This is because CUMUL is a statistical feature and Direction Sequence is a per-packet feature, both of which are easily obfuscated by WF defenses [19]. CIF dynamically adjusts time intervals to align the traffic distributions of Tor traffic, resulting in a lower MMD loss on all defenses compared to TAM. We also measure the MMD loss for all pairwise combinations of different network conditions. Figure 9 shows that CIF also achieves the lowest MMD loss across all network conditions, with an average of 0.44, while CUMUL and Direction Sequence exhibit a significantly higher MMD loss, with an average of 2.44 and 1.24, respectively.

Effectiveness of RobsuAugment. We measure the similarity between augmented and target trace representations using normalized Euclidean distance (i.e., Min-Max scaling) [25, 33], where augmented trace representations are generated by RobustAugment according to the original ones, while the target trace representations represent the corresponding target domain. A lower distance indicates higher similarity, illustrating better transferability across network conditions. We compare RobustAugment with NetAugment, a data augmentation strategy proposed by the SOTA WF attack framework NetCLR [2]. We use the D_1 dataset as the baseline, and leverage RobustAugment and NetAugment to generate augmented trace representations under different network conditions. We compute the similarity between these augmented representations and the target trace representations derived from the collected datasets

Table 8: The Euclidean distance between the augmented trace representations generated by NetAugment (NA) and RobustAugment (RA) and the target trace representation.

Network Conditions	Defense	NA	RA
Guard Relays	Undefended	0.37	**0.22**
	WTF-PAD	0.56	**0.22**
	Front	0.63	**0.23**
Times	Undefended	0.42	**0.18**
	WTF-PAD	0.62	**0.18**
	Front	0.63	**0.20**
Browsers	Undefended	0.33	**0.31**
	WTF-PAD	0.47	**0.32**
	Front	0.46	**0.32**

under three network conditions in three scenarios, i.e., without any defense, and with WTF-PAD [16] and with Front [8].

Table 8 shows that the Euclidean distance of RobustAugment under different network conditions is much lower than that of NetAugment. RobustAugment exhibits relatively stable Euclidean distances on both undefended and defended traces, whereas the average distance of NetAugment is 0.50, showing a significant increase particularly when dealing with defended traces. The reason is that NetAugment leveraging Direction Sequence as the trace representation is more easily obfuscated by WF defenses.

Impact on attack accuracy. We next evaluate the impact of three modules on the attack accuracy of Swallow. To evaluate the effectiveness of Swallow on pre-training datasets of different scales, we conduct experiments in two settings, i.e., pre-training on D_1 and fine-tuning on D_2 ($D_1 \rightarrow D_2$), and pre-training on DF95 and fine-tuning on D_1 (DF95 $\rightarrow D_1$). These experiments are conducted on WTF-PAD and Front, with the number of labeled traffic instances for fine-tuning fixed at $N = 5$. The results are shown in Table 9.

Module #1: Consistent Feature Representation. To explore the impact of CIF on the attack accuracy of Swallow, we replace it with CUMUL and TAM. The trace representation of NetCLR [2] does not include packet timing information of packet, while Swallow requires timing information. As a result, the Direction Sequence cannot be adapted to Swallow. The experimental results in Table 9 reveal that CUMUL and TAM cause average performance drops of around 40% and 6%, respectively. This is because CIF is more robust under different WF defenses and network conditions, as demonstrated in Table 7 and Figure 9.

Module #2: Robust Data Augmentation. To explore the impact of RobustAugment on the attack accuracy, we replace it with random insertion of Gaussian noise and NetAugment during the pre-training phase. The results show a significant decrease in the attack accuracy of model, with an average drop of around 40% on WTF-PAD traces. This is because RobustAugment can more effectively simulate the variations in defended traces, as illustrated in Table 8.

Module #3: Few-Shot Website Identification. We use ResNet18 [12] as the backbone for pre-training when designing Swallow. The SOTA WF attacks [2, 27, 32] mainly use two other CNN-based model architectures, which we refer to as the DF Backbone and RF Backbone. We replace the pre-trained backbone with the DF Backbone and RF Backbone. Compared to using ResNet18 as the pre-trained backbone, both DF-Backbone and RF-Backbone show an approximately 10% performance decline on WTF-PAD traces. This is because ResNet18 has a more complex network architecture [12], allowing it to better learn how to extract consistent features from the trace representation during the pre-training phase.

Table 9: Ablation study of key components in Swallow.

Category	Variation	$D_1 \rightarrow D_2$		DF95 $\rightarrow D_1$	
		WTF-PAD	Front	WTF-PAD	Front
Full	-	**83.33**	**66.59**	**69.80**	**60.27**
Representation	CUMUL	34.96	12.10	42.46	16.98
	TAM	78.02	61.42	62.73	53.37
Augmentation	Gaussian noise	31.35	30.58	35.33	30.43
	NetAugment	34.96	12.10	32.77	13.17
Pre-training Backbone	DF-Backbone	74.23	58.68	57.83	41.45
	RF-Backbone	73.21	65.69	63.50	53.74
Framework	SimCLR	61.78	64.57	66.92	53.74

We adopt BYOL [11] for self-supervised pre-training, while the NetCLR uses SimCLR [5]. To explore the performance difference, we replace BYOL with SimCLR. The pre-training datasets D_1 and DF95 contain 10,000 and 95,000 traffic instances. The attack accuracy of SimCLR is on average 10% lower on average than BYOL. The reason is that BYOL is well-suited for capturing the long-term data distributions of CIF [11]. Moreover, BYOL does not consider negative samples during pre-training, resulting in higher training overhead compared to SimCLR. The training time of BYOL per epoch on DF95 is only 79.51s, while SimCLR takes 228.46s, respectively.

7 Discussion

In this section, we discuss the limitations of the proposed WF attack and potential directions for future work.

Low attack accuracy against SOTA defense. We comprehensively evaluate the performance of Swallow across various scenarios in Section 6. The results demonstrate that Swallow achieves higher attack accuracy compared to other WF attacks, particularly when dealing with defended traces. However, the attack accuracy of Swallow remains relatively low against some of the existing defenses, such as the obfuscation-based defense Palette [28]. To address this limitation and further improve performance against such defenses, we can incorporate additional features that are both resistant to WF defense obfuscation and highly discriminative among websites.

Evaluation on real-world deployed defenses. Following previous work [27, 28, 31], all WF defenses evaluated in this work are simulated on undefended dataset. Their performance and overhead may be different when being implemented in the real world, especially for the defenses that delay packets [4, 9, 13, 28]. In future work, we will evaluate the attack performance of Swallow against these defenses in real-world scenarios.

Multi-tab browsing and webpage fingerprinting. In this work, we focus on identifying the websites users visit. Similar to previous studies [2, 27, 31], we assume that users visit one website at a time and only visit the homepage. However, in more realistic scenarios, users may simultaneously visit multiple websites and navigate beyond the homepage. These scenarios correspond to two key problems: multi-tab browsing (MTB) [14] or webpage fingerprinting (WPF) [37]. We can integrate Swallow into existing MTB and WPF

frameworks to enhance them. For example, we can enhance the robustness of MTB across different network conditions by utilizing CIF, and improve WPF via simulated webpage traffic variations under different network conditions. We leave these as future work.

8 Conclusion

In this paper, we proposed Swallow, a robust and transferable WF attack. Swallow can quickly transfer to new network conditions while demonstrating superior robustness against WF defenses compared to SOTA WF attacks. Specifically, we proposed a novel trace representation CIF, which aligns traffic distributions across different network conditions. We designed three data augmentation algorithms to expand the training dataset and simulate potential variations under various network conditions. We conducted extensive experiments to provide a comparison between Swallow and the SOTA WF attacks. The results from various scenarios demonstrate the superiority of Swallow over other WF attacks. In future work, we will improve the attack accuracy on SOTA defense and evaluate WF attacks against real-world deployed defenses.

Acknowledgment

This work is partially supported by National Key R&D Program of China with No. 2023YFB2703800, NSFC Projects with Nos. U23A20304, 62222201, and 62132011, and Beijing Natural Science Foundation with No. M23020. (Corresponding author: Meng Shen.)

References

[1] 2025. Vultr. https://www.vultr.com/
[2] Alireza Bahramali, Ardavan Bozorgi, and Amir Houmansadr. 2023. Realistic website fingerprinting by augmenting network traces. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1035–1049.
[3] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2018. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *arXiv preprint arXiv:1802.10215* (2018).
[4] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 227–238.
[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.
[6] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. 2020. Trafficsliver: Fighting website fingerprinting attacks with traffic splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1971–1985.
[7] Roger Dingledine, Nick Mathewson, Paul F Syverson, et al. 2004. Tor: The second-generation onion router.. In *USENIX security symposium*, Vol. 4. 303–320.
[8] Jiajun Gong and Tao Wang. 2020. Zero-delay lightweight defenses against website fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)*. 717–734.
[9] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. 2022. Surakav: Generating realistic traces for a strong website fingerprinting defense. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1558–1573.
[10] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *The Journal of Machine Learning Research* 13, 1 (2012), 723–773.
[11] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. 2020. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems* 33 (2020), 21271–21284.
[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[13] James K Holland and Nicholas Hopper. 2020. RegulaTor: A straightforward website fingerprinting defense. *arXiv preprint arXiv:2012.06609* (2020).
[14] Zhaoxin Jin, Tianbo Lu, Shuang Luo, and Jiaze Shang. 2023. Transformer-based Model for Multi-tab Website Fingerprinting Attack. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1050–1064.
[15] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. 263–274.
[16] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. 2016. Toward an efficient website fingerprinting defense. In *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I 21*. Springer, 27–46.
[17] Phuc H Le-Khac, Graham Healy, and Alan F Smeaton. 2020. Contrastive representation learning: A framework and review. *Ieee Access* 8 (2020), 193907–193934.
[18] Shuai Li, Huajun Guo, and Nicholas Hopper. 2018. Measuring information leakage in website fingerprinting attacks and defenses. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1977–1992.
[19] Shuai Li, Huajun Guo, and Nicholas Hopper. 2018. Measuring information leakage in website fingerprinting attacks and defenses. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1977–1992.
[20] Nate Mathews, James K Holland, Se Eun Oh, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2023. Sok: A critical evaluation of efficient website fingerprinting defenses. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 969–986.
[21] Perry Mike, Clark Erinn, Murdoch Steven, and Koppen Georg. 2018. The Design and Implementation of the Tor Browser [DRAFT]. https://2019.www.torproject.org/projects/torbrowser/design/.
[22] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale.. In *NDSS*, Vol. 1. 23477.
[23] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2018. Tranco: A research-oriented top sites ranking hardened against manipulation. *arXiv preprint arXiv:1806.01156* (2018).
[24] The Tor Project. 2024. https://metrics.torproject.org/.
[25] Yuqi Qing, Qilei Yin, Xinhao Deng, Yihao Chen, Zhuotao Liu, Kun Sun, Ke Xu, Jia Zhang, and Qi Li. 2024. Low-Quality Training Data Only? A Robust Framework for Detecting Encrypted Malicious Network Traffic. In *NDSS*.
[26] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2019. Tik-tok: The utility of packet timing in website fingerprinting attacks. *arXiv preprint arXiv:1902.06421* (2019).
[27] Meng Shen, Kexin Ji, Zhenbo Gao, Qi Li, Liehuang Zhu, and Ke Xu. 2023. Subverting website fingerprinting defenses with robust traffic representation. In *32nd USENIX Security Symposium (USENIX Security 23)*. 607–624.
[28] Meng Shen, Kexin Ji, Jinhe Wu, Qi Li, Xiangdong Kong, Ke Xu, and Liehuang Zhu. 2024. Real-Time Website Fingerprinting Defense via Traffic Cluster Anonymization. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 263–263.
[29] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, and Xiaojiang Du. 2021. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Transactions on Information Forensics and Security* 16 (2021), 2367–2380.
[30] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data* 6, 1 (2019), 1–48.
[31] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 1928–1943.
[32] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. 2019. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1131–1148.
[33] Chao Wang, Alessandro Finamore, Pietro Michiardi, Massimo Gallo, and Dario Rossi. 2024. Data augmentation for traffic classification. In *International Conference on Passive and Active Network Measurement*. Springer, 159–186.
[34] Tao Wang. 2020. High Precision Open-World Website Fingerprinting. In *2020 IEEE Symposium on Security and Privacy (SP)*. 152–167.
[35] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)*. 143–157.
[36] Renjie Xie, Jiahao Cao, Enhuan Dong, Mingwei Xu, Kun Sun, Qi Li, Licheng Shen, and Menghao Zhang. 2023. Rosetta: Enabling Robust {TLS} Encrypted Traffic Classification in Diverse Network Environments with {TCP-Aware} Traffic Augmentation. In *32nd USENIX Security Symposium (USENIX Security 23)*. 625–642.
[37] Xiyuan Zhao, Xinhao Deng, Qi Li, Yunpeng Liu, Zhuotao Liu, Kun Sun, and Ke Xu. 2024. Towards Fine-Grained Webpage Fingerprinting at Scale. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 423–436.

Table 10: Accuracy of the SOTA WF attacks against defenses with various labeled traffic instances.

N	WF Attacks	Accuracy (%)								
		Undefended	WTF-PAD [16]	Front [8]	Surakav [4]	RegulaTor [13]	Palette [28]	Tamaraw [4]	UniDef	TrafficSliver [6]
10	DF [31]	70.47	18.64	10.48	8.13	5.17	4.13	7.66	2.00	24.11
	Tik-Tok [26]	89.88	77.66	29.98	39.75	5.04	4.42	8.53	2.36	19.95
	Var-CNN [3]	68.49	12.20	4.73	38.27	1.20	1.38	4.24	1.40	7.38
	RF [27]	86.46	73.06	67.15	44.08	35.48	11.31	8.87	3.89	53.97
	TF [32]	91.35	76.91	28.95	32.74	2.37	2.72	8.21	1.46	17.51
	NetCLR [2]	91.54	82.75	28.98	28.90	2.38	2.87	8.75	1.99	16.26
	Swallow	**91.68**	**87.42**	**78.20**	**50.18**	**40.97**	**11.83**	**9.88**	**4.07**	**57.81**
15	DF [31]	79.58	34.40	17.39	9.23	5.32	4.09	7.80	2.01	33.93
	Tik-Tok [26]	90.40	82.52	44.28	45.34	6.89	4.57	8.40	2.10	32.07
	Var-CNN [3]	73.23	28.45	12.46	39.23	2.21	2.13	5.14	1.98	8.87
	RF [27]	90.52	81.43	77.07	52.97	47.92	13.02	10.33	3.84	60.24
	TF [32]	91.57	77.46	35.39	34.65	2.46	2.86	8.58	2.11	18.88
	NetCLR [2]	91.68	85.66	38.45	32.48	2.45	3.24	8.89	2.38	16.78
	Swallow	**92.70**	**88.90**	**83.24**	**55.89**	**49.20**	**13.98**	**11.01**	**4.12**	**63.87**
20	DF [31]	83.20	43.15	19.72	10.90	5.42	3.72	8.12	2.25	42.17
	Tik-Tok [26]	92.22	84.07	52.67	47.45	7.50	4.67	8.72	2.64	38.45
	Var-CNN [3]	77.10	33.60	19.65	41.58	2.30	2.32	4.60	1.43	9.68
	RF [27]	91.92	83.40	79.77	55.85	50.52	13.49	11.38	4.02	65.25
	TF [32]	92.03	78.68	40.58	36.22	2.51	3.15	8.14	2.62	19.06
	NetCLR [2]	92.10	87.01	40.58	36.44	2.60	3.46	9.07	2.54	19.18
	Swallow	**92.93**	**90.45**	**85.84**	**59.56**	**52.22**	**14.28**	**11.52**	**4.23**	**66.66**

Algorithm 4: Trace Aggregation Algorithm

Input: $CIF = [u, d]$, loading time t, interval duration s, remove probability r_{remove}, increase probability $r_{increase}$
Output: CIF_{aug}

Let len = length(u);
Initialize $CIF_{aug} = [[], []]$, $i = 0$;
while $i < len$ *and* $i \cdot s < t$ **do**
  i+ = 1;
  foreach $dir \in \{u, d\}$ **do**
    if *random()* < r_{remove} **then**
      continue
    end
    if *random()* > $r_{increase}$ **then**
      $ts = CIF[dir][i]$;
      $ts \times = (1 + [0, 1])$;
      $CIF_{aug}[dir].append(ts)$;
    end
  end
end
return CIF_{aug};

Algorithm 5: Trace Flatten Algorithm

Input: $CIF = [u, d]$, loading time t, interval duration s, insert probability r_{insert}, shrink probability $r_{decrease}$, number of windows for averaging W_{insert}
Output: CIF_{aug}

Let len = length(u);
Initialize $CIF_{aug} = [[], []]$, $i = 0$;
while $i < len$ *and* $i \cdot s < t$ **do**
  foreach $dir \in \{u, d\}$ **do**
    if *random()* < r_{insert} **then**
      ts_{insert} = avg over W_{insert};
      $CIF_{aug}[dir].append(ts_{insert})$;
    end
    else
      if *random()* > $r_{decrease}$ **then**
        $ts = CIF[dir][i]$;
        $ts \times = (1 + [-1, 0])$;
        $CIF_{aug}[dir].append(ts)$;
        i+ = 1;
      end
    end
  end
end
return CIF_{aug};

Algorithm 3: Trace Fluctuation Algorithm

Input: $CIF = [u, d]$, loading time t, interval duration s, padding probability $r_{padding}$, modification probability r_{modify}, window size for averaging W_{modify}
Output: CIF_{aug}

Initialize CIF_{aug} with zeros and index $i = 0$;
Let len = length(u);
while $i < len$ *and* $i \cdot s < t$ **do**
  i+ = 1;
  foreach $dir \in \{u, d\}$ **do**
    $ts = CIF[dir][i]$;
    if $ts == 0$ **then**
      if *random()* > $r_{padding}$ **then**
        ts = avg over W_{modify};
      end
    else
      if *random()* > r_{modify} **then**
        $ts \times = (1 + [-1, 1])$;
      end
    end
    $CIF_{aug}[dir][i] = ts$;
  end
end
return CIF_{aug};

A Details of Augmentation Algorithms

In this section, we provide the detailed implementation of the three proposed algorithms: *Trace Fluctuation*, *Trace Aggregation*, and *Trace Flatten*. These algorithms are designed to simulate various network conditions and are presented in Algorithm 3, Algorithm 4, and Algorithm 5, respectively.

B Evaluation on Different Guard Relays

The attack performance of the SOTA WF attacks with various labeled traffic instances ($N = 10, 15, 20$) is shown in Table 10, to complement the results presented in Section 6.2. Swallow still achieves the best performance under different labeled traffic instances.