Adaptive Fingerprinting: Website Fingerprinting over Few Encrypted Traffic

Chenggang Wang
University of Cincinnati
Cincinnati, OH, USA
wang2c9@mail.uc.edu

Jimmy Dani
University of Cincinnati
Cincinnati, OH, USA
danijy@mail.uc.edu

Xiang Li
University of Cincinnati
Cincinnati, OH, USA
li5xi@ucmail.uc.edu

Xiaodong Jia
University of Cincinnati
Cincinnati, OH, USA
jiaxg@ucmail.uc.edu

Boyang Wang
University of Cincinnati
Cincinnati, OH, USA
boyang.wang@uc.edu

ABSTRACT

Websitefi ngerprinting attacks can infer which website a user visits over encrypted network traffic. Recent studies can achieve high accuracy (e.g., 98%) by leveraging deep neural networks. However, current attacks rely on enormous encrypted traffic data, which are time-consuming to collect. Moreover, large-scale encrypted traffic data also need to be recollected frequently to adjust the changes in the website content. In other words, the bootstrap time for carrying out websitefi ngerprinting is not practical. In this paper, we propose a new method, named Adaptive Fingerprinting, which can derive high attack accuracy over few encrypted traffic by leveraging adversarial domain adaption. With our method, an attacker only needs to collect few traffic rather than large-scale datasets, which makes websitefi ngerprinting more practical in the real world. Our extensive experimental results over multiple datasets show that our method can achieve 89% accuracy over few encrypted traffic in the closed-world setting and 99% precision and 99% recall in the open-world setting. Compared to a recent study (named Triplet Fingerprinting), our method is much more efficient in pre-training time and is more scalable. Moreover, the attack performance of our method can outperform Triplet Fingerprinting in both the closed-world evaluation and open-world evaluation.

CCS CONCEPTS

• **Security and privacy** → **Network security**.

KEYWORDS

Encrypted traffic, transfer learning, adversarial domain adaption

ACM Reference Format:
Chenggang Wang, Jimmy Dani, Xiang Li, Xiaodong Jia, and Boyang Wang. 2021. Adaptive Fingerprinting: Website Fingerprinting over Few Encrypted Traffic. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy (CODASPY '21), April 26–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3422337.3447835

1 INTRODUCTION

In websitefi ngerprinting [4, 5, 9, 13, 15, 23, 27, 29, 31, 35, 43], an attacker eavesdrops encrypted traffic and infers which website a user visits in secure communication protocols, such as Tor. Website fingerprinting is often formulated as a supervised learning problem, where an attacker collects a large-scale dataset and trains a classifier leveraging machine learning. Despite the substantial process, especially with deep neural networks, studies on websitefingerprinting still face challenges. One of the primary challenges is that current websitefi ngerprinting methods rely on collecting enormous traffic data, which is extremely time-consuming [36]. In other words, the bootstrap time for carrying out websitefi ngerprinting is not practical in the real world.

For instance, it takes more than 30 days to collect a large-scale dataset for analyzingfi ngerprints of website traffic [31, 35, 36]. Even data collection time can be significantly reduced by running multiple computers or virtual machines in parallel, obtained traffic data are easily outdated due to the content updates on websites. For example, studies [31, 44] have shown that if test traffic data are collected more than 14 days later than training traffic data, the attack accuracy will drop significantly. As a result, an attacker will need to recollect data frequently. To make it even worse, if there is any inconsistency in terms of data collection setting (e.g., versions of software, operating systems, network protocols, etc.) between an attacker and a target user, an attacker has to match the setting and recollects extensive data accordingly.

In this paper, we propose a new method, referred to as *Adaptive Fingerprinting*, which can perform websitefi ngerprinting and derive high attack accuracy over only few encrypted traffic. In other words, our method does not need to collect large-scale traffic data, which reduces the bootstrap time for websitefi ngerprinting attacks and makes the attacks practical in the real world. Our main idea is to leverage transfer learning, more specifically, *adversarial domain adaption* [10, 39], to transfer knowledge learned from an existing large-scale dataset to the classification over a dataset with few traffic (e.g., no more than 20 traces per monitored website). Following the definitions in the literature of transfer learning, we denote this existing large-scale dataset as a *source dataset* and this dataset with few traffic as a *target dataset* in this paper. The main contributions of this paper can be summarized below:

- In Adaptive Fingerprinting (AF), we leverage adversarial domain adaption, more specifically, a domain adversarial network [10, 39], to learn a Feature Extractor over one or multiple source datasets by formulating a *minimax game* [12] between a Feature Extractor and a Domain Discriminator. A Feature Extractor or a Domain Discriminator is, in essence, a deep neural network. The learned Feature Extractor is extracted and attached with a traditional machine learning classifier (e.g., k-nearest neighbor) to carry out the classification over a target dataset.
- In our closed-world evaluation, our experimental results over multiple datasets show that our method can achieve high accuracy over a target dataset, which has no more than 20 traces per monitored website. For instance, our method can achieve over 89% accuracy over 100 monitored websites. In the open-world evaluation, our method can achieve 99% precision and 99% recall.
- Compared to a previous method, named Triplet Fingerprinting [36], which also performs websitefi ngerprinting over few traffic, our method is much more efficient in pre-training time (i.e., the time to train a feature extractor from a source dataset) and is more scalable if there are more data available in a source dataset. Our method can outperform Triplet Fingerprinting in the closed-world evaluation, except when there is only 1 trace per monitored website in a target dataset.

Reproducibility. The source code and datasets of this study are publicly available and can be found at [3].

2 BACKGROUND

System and Threat Model. In this paper, we consider a system model including three parties, a user, an attacker and a web server. Afi gure of the model is illustrated in Fig. 1. This user connects to the web server through Tor relays by using the Tor protocol. The network traffic between the user and a web server is encrypted. We assume there is an attacker, who is able to eavesdrop encrypted traffic between a user and thefi rst Tor relay. The goal of this attacker is to infer which website this user visits by analyzing the size and direction of encrypted packets. The system model we consider in this paper is the same as previous studies in websitefingerprinting.

By following the assumptions in the existing studies [5, 27, 29, 31, 35], we assume that an attacker does not know the secret key to decrypt packets. Moreover, we assume that this attacker is passive and does not drop or inject packets. We assume that a user visits one website each time. There are minimal background traffic from other applications or websites. A *traffic trace* is a sequence of incoming and outgoing network packets related to one website visit.

Binary Format of Traffic Traces. As Tor implementsfi xed-length packets, named cells [31], to transmit data in the Tor protocol, we use the binary format to represent each traffic trace as in previous studies. Specifically, given a traffic trace, we only keep the direction of each packet. We use +1 to represent an outgoing packet (to a website) and -1 to indicate an incoming packet (from a website). Each traffic trace, in essence, is a vector of +1s and -1s. To

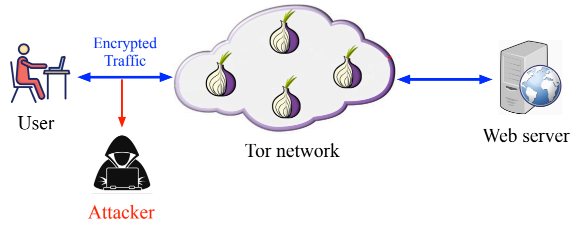


Figure 1: The system and threat model. We assume that an attacker can eavesdrop encrypted traffic between a user and thefi rst Tor relay.

use these vectors as the inputs of neural networks, we also keep the same length for all the vectors by trimming or padding 0s at the end of each vector as in previous studies [31, 35].

Closed-World Setting and Open-World Setting. Websitefingerprinting can be evaluated in two settings, including the closed-world setting and open-world setting. We examine both settings in this paper. In a closed-world setting, we assume that a user only visits a set of monitored websites and the attacker knows this set of monitored websites. Given an unlabeled traffic trace, an attacker infers which specific website it belongs to. In an open-world setting, we assume that a user can also visit unmonitored websites in addition to the set of monitored websites. Given an unlabeled traffic trace, an attacker infers whether this trace is associated with monitored websites or unmonitored websites.

Evaluation Metrics. Both the closed-world evaluation and open-world evaluation can be formulated as classification problems, where the closed-world evaluation carries out a multi-class classification while the open-world evaluation performs a binary classification. Accuracy is used as a metric to measure the attack performance in the closed-world evaluation. Precision, recall, and precision-recall curves are utilized in the open-world evaluation.

For the open-world evaluation, we apply the *standard model* used in previous studies [35, 36]. Specifically, all the traffic traces of unmonitored websites are considered as a single class, which is added to the classifier obtained in the closed-world evaluation as an additional class. This classifier is re-trained with traces from monitored websites and unmonitored websites. During the test, given an unlabeled traffic trace, if the highest confidence of this classifier belongs to one of the monitored websites and this confidence is greater than a threshold, this trace is considered as a trace associated with monitored websites. Otherwise, it is considered as a trace related to unmonitored websites. The threshold can be tuned in experiments to obtain a higher precision or higher recall.

Our Goals in This Study. Our study formulates websitefingerprinting as a transfer learning problem. Specifically, we assume a large-scale dataset, referred to as a *source dataset*, is available but it was collected with different settings (e.g., different versions in software, hardware, and network protocols) in the past. In addition, another dataset, referred to as a *target dataset*, is collected based on the latest setting of a target user. However, this target dataset only has few labeled traffic (to be more specifically, less than 20 traces per website in this paper).

We have *two specific goals* in this study. First, we would like to perform websitefi ngerprinting over few traffic of a target dataset with high accuracy by taking advantage of the large amount of traffic from a source dataset. Second, we aim to render efficient

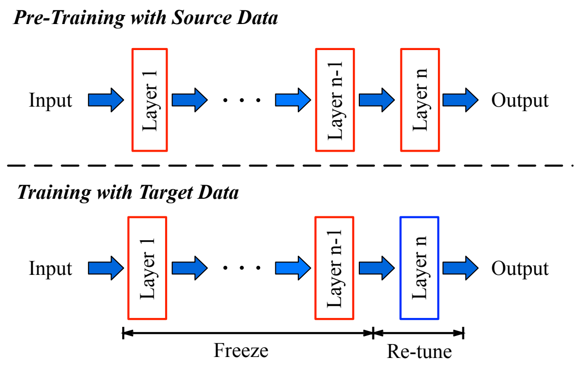


Figure 2: Transfer learning withfine-tuning

running time when performing transfer learning in websitefin-gerprinting. The *overarching goal* of our study is to minimize the bootstrap time, including data collection time and classifier training time, and make websitefi ngerprinting more practical.

3 TRANSFER LEARNING OVER ENCRYPTED TRAFFIC

3.1 Transfer Learning

Supervised learning is data-hungry and does not perform well when a dataset consists of few labeled data. Transfer learning [28, 49] is able to overcome this limitation. Specifically, given a source dataset and a target dataset, where the source dataset has a large amount of labeled samples and the target dataset has few labeled samples, transfer learningfi rst learns the knowledge from the source data, and then transfer the knowledge to perform the classification task over the target data. The knowledge often indicates tuned hyperparameters of a neural network or learned feature spaces. Transfer learning performs well when the source classification task is similar to the target classification task. For example, the learned knowledge of recognizing cars can be transferred to identify trucks.

A transfer learning method, in general, consists of three steps, including *pre-training*, *training*, and *testing*. In pre-training, the knowledge of a source dataset is learned. In the training step, the learned knowledge is transferred by leveraging the training data of the target dataset. In the testing step, a classifier reports results over the test data of the target dataset.

3.2 Fine-Tuning

Fine-tuning [47] is one of the simplest methods in transfer learning. In the pre-training step, this method trains a neural network over a source dataset. In the training step, this method freezes most of the layers in the neural network obtained from the pre-training step and tunes hyperparameters of the last few layers using training data from the target dataset. Finally, with the re-tuned neural network, this method derives the results with test data of the target dataset. The process offi ne-tuning is described in Fig. 2.

Fine-tuning often performs well when the source domain and target domain are very similar. This is because the shallow layers obtained from the pre-training step can extract general features that are likely shared by both the source domain and target domain [47]. Re-tuning the hyperparameters only in the last 1 or 2 layers with data from the target dataset can adjust the neural network

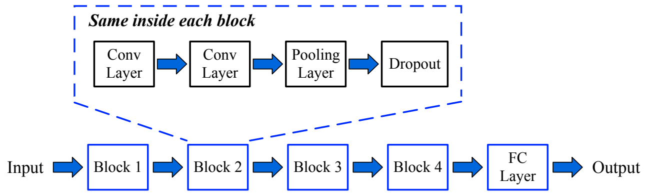


Figure 3: The structure of the DF model. FC layer stands for Fully-Connected layer.

to be more accurate for the target domain. Besides, the number of hyperparameters that need to be re-tuned is much smaller than re-training the entire neural network, which is more convenient for limited data from the target dataset.

Fine-Tuning over Encrypted Traffic. A previous study [36] reported the performance offi ne-tuning over encrypted traffic in the context of websitefi ngerprinting. This studyfi rst trained a neural network, named DF (Deep Fingerprinting) model [35], by using a source dataset. Next, itfi ne-tuned the DF model with a target dataset. The DF model is one of the most effective models for classifying encrypted traffic with supervised learning. The structure of the DF model is highlighted in Fig. 3.

3.3 Triplet Networks

The triplet network [30, 32], inspired by the Siamese network [21, 38], contains three parallel identical sub-networks sharing the same weights and hyperparameters. An input of a triplet network is denoted as a *triplet*, which consists of an anchor sample A, a positive sample P and a negative sample N. Each sub-network takes only one type of samples as inputs. For instance, all the anchor samples A are the inputs to one sub-network and this sub-network does not include any positive samples or negative samples in its inputs. Each triplet is selected from the source dataset, either randomly or with some mining strategy [36].

The details of transfer learning using a triplet network is described in Figure 4. In the pre-training phase, a triplet network is trained to learn an embedding of source data. The goal is to train the sub-networks such that the distance between anchor samples and positive samples is smaller than the distance between anchor samples and negative samples in an embedded space. The triplet loss is leveraged to measure the training loss of the triplet network [30, 32]. Specifically, given a triple (A, P,N), the lost function is defined as

$$\mathcal{L}(A, P,N) = \mathtt{max}(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha, 0) \quad (1)$$

where $f(\cdot)$ is the embedding, α is a margin between positive and negative samples. Besides L2 distance presented in the equation above, cosine distance can also be utilized [30, 32].

In the training phase, the sub-network from the triplet network is extracted and utilized as a feature extractor. One classifier, referred to as target classifier, is attached to this feature extractor. The parameters of this target classifier are trained using the target training data. All the hyperparameters and weights in the sub-network remain the same. Last, the results are reported over the target test data with the sub-network and the trained classifier.

Triplet Fingerprinting. Triplet Fingerprinting (TF) [36] examines websitefi ngerprinting by leveraging triplet networks. In

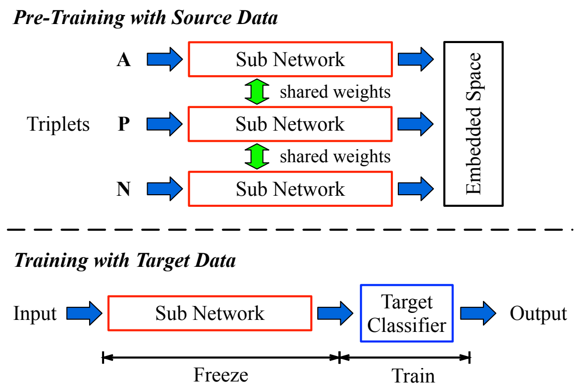


Figure 4: Transfer learning with a triplet network.

Triplet Fingerprinting, anchor samples and positive samples are selected from the same classes while negative samples are chosen from different classes compared to anchor samples. The authors leveraged the DF model [35] as the sub-network in a triplet network. A k-nearest neighbor (k-NN) classifier is used as the target classifier during the training over the target dataset.

4 ADAPTIVE FINGERPRINTING

Wefi rst present the main idea of adversarial domain adaption. Next, we discuss how we address the specific challenges of adversarial domain adaption in the context of websitefingerprinting.

4.1 Adversarial Domain Adaption

Domain adaption [40] is one of the transfer learning approaches. Domain adaption addresses the domain shift problem (i.e., the discrepancy between a source dataset and a target dataset) [37] by mapping both source data and target data into a domain-invariant feature space. Traditional domain adaption methods [24, 40] often minimize the discrepancy by measuring the distance with Maximum Mean Discrepancy.

Adversarial Domain Adaption [10, 39] leverages a *domain adversarial network* to learn a domain-invariant feature space. It leverages the idea of *generative adversarial learning* [12] and outperforms the traditional ones relying on Maximum Mean Discrepancy. Assume there is a source dataset and a target dataset, the structure of a domain adversarial network consists of a Feature Extractor F, a Domain Discriminator D, and a Source Classifier C as shown in Fig. 5. The Feature Extractor, Domain Discriminator, or Source Classifier, in essence, is a neural network. The parameters of the Feature Extractor, Domain Discriminator and Source Classifier can be represented as θ_F, θ_D, and θ_C respectively.

Note that existing adversarial domain adaption methods often assume that the source dataset and target dataset have the same label space (i.e., the source dataset and target dataset share the same set of class labels). In addition, existing methods also assume that *the target dataset consists of a large amount of unlabeled data rather than few labeled data.* We will address the scenarios where the target dataset has a different label space and consists of few labeled data in our method presented in the next subsection.

During the training of a domain adversarial network, the Feature Extractor takes source data and target data as inputs and aims to output domain-invariant features, which are difficult for the

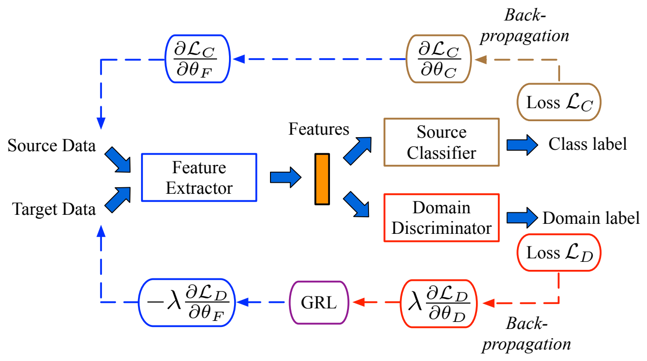

Back-propagation
$\frac{\partial \mathcal{L}_C}{\partial \theta_F}$
$\frac{\partial \mathcal{L}_C}{\partial \theta_C}$
Loss $\mathcal{L}_C$
Source Data
Target Data
Feature Extractor
Features
Source Classifier
Class label
Domain Discriminator
Domain label
Loss $\mathcal{L}_D$
$-\lambda\frac{\partial \mathcal{L}_D}{\partial \theta_F}$
GRL
$\lambda\frac{\partial \mathcal{L}_D}{\partial \theta_D}$
Back-propagation

Figure 5: The structure of a domain adversarial network [10]. GRL stands for Gradient Reversal Layer.

Domain Discriminator to distinguish. The Domain Discriminator, on the other hand, aims to distinguish whether an output of the Feature Extractor is produced by data from the source dataset or data from the target dataset. The Source Classifier aims to minimize its loss on predicting the correct class label of source data with the outputs produced by the Feature Extractor.

In other words, given the loss function $\mathcal{L}$ of the entire domain adversarial network, the Feature Extractor and Source Classifier aim to minimize the loss $\mathcal{L}$ while the Domain Discriminator aims to maximize the loss $\mathcal{L}$. The training objective of the entire network is to achieve the following:

$$\begin{aligned}\left(\hat{\theta}_F, \hat{\theta}_C\right) &= \underset{\theta_F,\theta_C}{\operatorname{argmin}}\, \mathcal{L}(\theta_F, \hat{\theta}_D, \theta_C)\\ \hat{\theta}_D &= \underset{\theta_D}{\operatorname{argmax}}\, \mathcal{L}(\hat{\theta}_F, \theta_D, \hat{\theta}_C)\end{aligned} \tag{2}$$

where $\hat{\theta}_F$, $\hat{\theta}_D$, $\hat{\theta}_C$ are the optimal values of θ_F, θ_D, and θ_C respectively. The loss function $\mathcal{L}$ can be computed as

$$\mathcal{L}(\theta_F, \theta_D, \theta_C) = \mathcal{L}_C(\theta_F, \theta_C) - \lambda\mathcal{L}_D(\theta_F, \theta_D) \tag{3}$$

where $\mathcal{L}_C$ is the loss function of the Source Classifier, $\mathcal{L}_D$ is the loss function of the Domain Discriminator, and λ is a pre-defined trade-off parameter shaping features during learning [10]. The parameters of the entire network are updated through back-propagation, where the updates are operated as below:

$$\begin{aligned}\theta_F &= \theta_F - \alpha\left(\frac{\partial\mathcal{L}_C}{\partial\theta_F} - \lambda\frac{\partial\mathcal{L}_D}{\partial\theta_F}\right)\\ \theta_C &= \theta_C - \alpha\frac{\partial\mathcal{L}_C}{\partial\theta_C}\\ \theta_D &= \theta_D + \alpha\frac{-\lambda\partial\mathcal{L}_D}{\partial\theta_D}\end{aligned} \tag{4}$$

where α is the learning rate, and Gradient Reversal Layer (GRL) assigns negative signs (i.e., the ones before parameter λ) to the derivative of $\mathcal{L}_D$ with respect to θ_F and θ_D. Gradient Reversal Layer is introduced in order to evaluate the gradients of θ_F and θ_D in each back-propagation epoch [10].

After the training of a domain adversarial network, the Feature Extractor F and the Classifier C can be extracted out and directly used to perform classifications over target data.

4.2 Our Proposed Method

As we mentioned in the last subsection, existing adversarial domain adaptions assume the source dataset and the target dataset share the same label space. Unfortunately, these assumptions do not hold over encrypted traffic in the context of websitefingerprinting.

Challenges. Specifically, given two encrypted traffic datasets, the selection of the websites can be very different. For instance, one dataset can choose websites according to Alexa top websites [31, 35] while others can choose sensitive websites that are blocked by certain countries [43]. In addition, even the set of selected websites is exactly the same between a source dataset and a target dataset, the two datasets are often collected several months or years apart, where the traffic pattern of the same website can change dramatically due to the content changes on a website [31].

A Straightforward Solution. Given the target data are labeled in our study rather than unlabeled, adding an additional neural-network-based Target Classifier within an domain adversarial network described in Fig. 5 could be a potential way to mitigate the problem. This Target Classifier would be parallel to the Source Classifier. However, as the target dataset only has few labeled data , it may not be sufficient to derive a well-trained neural-network-based Target Classifier within a domain adversarial network.

Our Proposed Method. To address the challenges over encrypted traffic data, we propose a new method named Adaptive Fingerprinting (AF). The main idea is to leverage a domain adversarial network to learn domain-invariant features only. Then, our method extracts the feature extractor and attaches a traditional machine learning classifier, which is much easier to train with limited labeled data in the target dataset. Our method still consists of three phases, including pre-training, training, and testing. Depending whether there are multiple source datasets available, our method can be represented in the two following versions. We denote the two versions as *AF-SingleSource* and *AF-MultiSource* respectively.

Details of AF-SingleSource. AF-SingleSource (as illustrated in Fig. 6) is suitable for the cases where there is 1 source dataset (with a large amount of labeled data) and 1 target dataset (with few labeled data). The target dataset is divided into target training data and target test data. Specifically, in the pre-training phase, our method trains a domain adversarial network by taking the source dataset and target training data as inputs. Although these target training data are labeled, our method treats them as unlabeled in the pre-training phase. The domain adversarial network still consists of Feature Extractor, Domain Discriminator, and Source Classifier.

Next, in the training phase, our method extracts the trained Feature Extractor out and attaches a traditional machine learning classifier, which is utilized as a target classifier. Our method trains the parameters of this target classifier with target training data by freezing the Feature Extractor (except its last layer). Finally, in the testing phase, our method obtains results over target test data with this Feature Extractor and the target classifier.

Details of AF-MultiSource. If the source data and the target data in AF-SingleSource are significantly unbalanced, it will likely affect the training of the Domain Discriminator, which essentially affect the training of Feature Extractor and fail to derive domain-invariant features. To mitigate this limitation, our method can take multiple source datasets in the pre-training phase instead of using

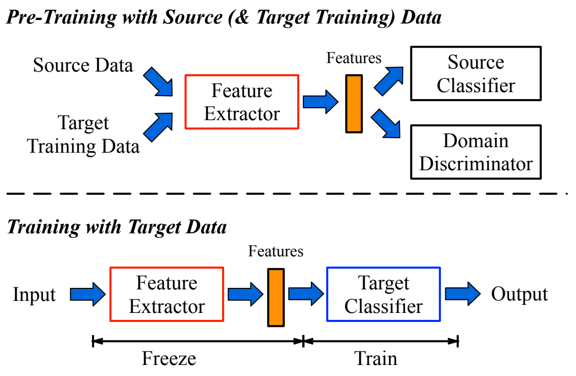


Figure 6: The process of our method AF-SingleSource.

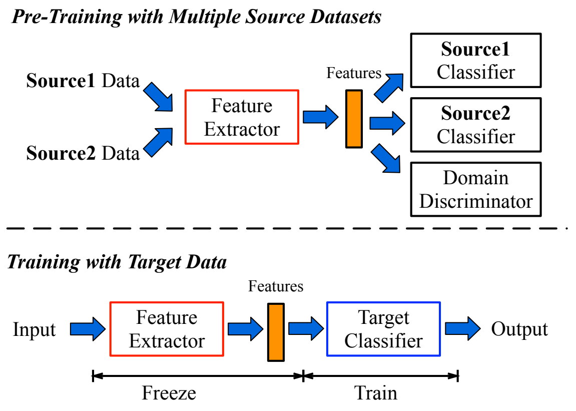


Figure 7: The process of our method AF-MultiSource.

target data as a part of the inputs to a domain adversarial network. The tradeoff is that we need to leverage multiple source datasets rather than one source dataset.

AF-MultiSource (as shown in Fig. 7) is suitable for the cases where there are multiple source datasets (each with a large amount of labeled data) and 1 target dataset (with few labeled data). The target data is still divided into target training data and target test data. In the pre-training phase, our method trains a domain adversarial network by taking multiple source datasets as inputs. The domain adversarial network consists of Feature Extractor, Domain Discriminator, and multiple Classifiers, where each source dataset is assigned one classifier. The training and testing phase remains the same as in AF-SingleSource.

Structures of Neural Networks in Our Method. For the Feature Extractor in our domain adversarial network, we leverage the DF model [35] presented in Fig. 3. For the Domain Discriminator in our domain adversarial network, it consists of 2 convolutional layers, 2 pooling layers and 1 fully-connected layer (with `softmax` as the activation function). For the Classifier in our domain adversarial network, we use 1 convolutional layer, 1 pooling layer and 1 fully-connected layer (with `softmax` as the activation function). The structures of the Discriminator and Classifier can be found in Fig. 8 and Fig. 9. If there are multiple source datasets, each source classifier will use the same structure as the Classifier illustrated in Fig. 9. For the target classifier, we use a k-nearest neighbor classifier.

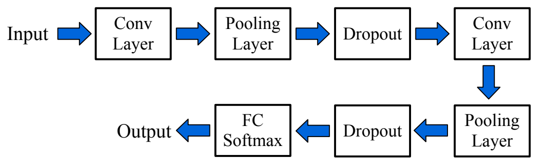


Figure 8: The Domain Discriminator used in our method.

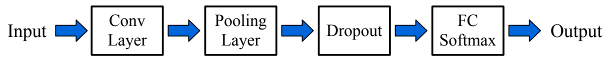


Figure 9: The Source Classifier used in our method.

5 PERFORMANCE EVALUATION

5.1 Datasets

In this study, we collected one new dataset (named AF dataset) and also leveraged three datasets from previous studies to examine the performance of our method. Each dataset includes Tor traffic traces from monitored websites (for closed-world evaluation) and unmonitored websites (for open-world evaluation). These datasets were collected with different settings and times. In addition, how the monitored websites and unmonitored websites were selected is also different among some of them.

Wang dataset [42]. This dataset was collected in 2013 by using Tor Browser 3.5. The monitored websites were selected from a list of sensitive sites blocked in three countries (China, the UK, and Saudi Arabia). The unmonitored websites were selected from Alexa top websites[1]. The 2 subsets of this dataset are summarized below:

- Wang100: It includes a set of 100 monitored websites. Each website has 90 traces.
- Wang9000: It includes a set of 9,000 non-monitored websites. Each website has 1 trace.

AWF dataset [31]. This dataset was collected in 2016 by utilizing Tor Browser 6.5. The monitored websites were chosen from Alexa top websites. The unmonitored websites were also selected from Alexa top websites (excluding the ones selected in monitored websites). The subsets we utilize in this paper are described below:

- AWF100: It includes a set of 100 monitored websites. Each website has 2,500 traces.
- AWF775: It includes a set of 775 monitored websites. Each website has 2,500 traces.
- AWF9000: It includes a set of 9,000 unmonitored websites. Each website has 1 trace.

DF dataset [35]. This dataset was collected in 2016 using Tor Browser 6.X. The monitored websites were selected from Alexa top websites. The unmonitored websites were chosen from Alexa top websites (excluding the ones in monitored websites).

- DF95: It includes a set of 95 monitored websites with 1,000 traces per website.
- DF9000: It includes a set of 9,000 unmonitored websites with 1 trace per website.

AF dataset (This paper). We collected this dataset from July to September in 2020 using Tor Browser 9.0. We utilized 5 virtual machines on campus, where each virtual machine has an identical configuration (Ubuntu 18.04 and 4 GB RAM). We used an open-source tool, named `tor-browser-crawler` [2], to collect encrypted Tor traffic. This tool was also used in previous studies [18] for Tor traffic collection.

The monitored websites were chosen from top 130 of Alexa websites. 300 traces per website were initially collected. As some of the websites block Tor traffic and their corresponding traffic contains a very small number of packets, we considered those traces invalid and removed them. After cleaning, we obtained 100 monitored websites with 250 traces per website. The unmonitored websites were selected from Alexa websites (top 130 to top 10300 websites). We collected 1 trace for each unmonitored website. After cleaning, we obtained 9,000 valid traces. The two subsets are summarized below:

- AF100: It includes a set of 100 monitored websites with 250 traces per website.
- AF9000: It includes a set of 9,000 unmonitored websites with 1 trace per website.

5.2 Experiment Settings

We implement neural networks and machine learning algorithms in Python. For neural networks, we use Keras as the front end and Tensorflow as the back end. We run all the experiments on a Linux machine with Ubuntu 18.04, 2.8 GHz CPU, 64 GB RAM, and a NIVIDA Titan RTX GPU. We perform 10-fold cross validations.

We leverage the DF model in pre-training for the three methods, includingfi ne-tuning, Triple Fingerprinting, and our method. We leverage the source code of the DF model and its tuned hyperparameters reported in [35] in our experiments. We train the DF model with 30 epochs in pre-training each time.

Implementation of Fine-Tuning. We pre-train the DF model with a source dataset and then tune the hyperparameters of the last layer of the DF model in the training step.

Implementation of Triplet Fingerprinting. We leverage the source code published by the authors of Triplet Fingerprinting [36] to implement it in our experiments. All the settings for Triplet Fingerprinting are the same as the ones described in [36]. Specifically, we use the DF model as the sub-network in a triplet network. We use cosine distance as the distance metric to calculate the triplet loss. In addition, we use Semi-Hard-Negative mining strategy to select triplets from a source dataset. For the k-nearest neighbor classifier in Triplet Fingerprinting, we choose $k = N$, where N is the number of traces per class in target training data. We also use the Mean Embedded Vector of N samples to train k-nearest neighbor in the closed-world evaluation. Using the Mean Embedded Vector rather than N samples can lead to a higher accuracy as mentioned in [36]. More details can be found in [36].

Implementation of Our Method. As mentioned, we use the DF model as the Feature Extractor. The structures of Domain Discriminator and Classifier are described in Fig. 8 and Fig. 9. We tuned hyperparameters of Domain Discriminator and Source Classifier based on AWF100 (as the source dataset) and Wang100 (as the target dataset). Then we keep the same hyperparameters of Domain Discriminator and Source Classifier for all the experiments we have. Some of the key hyperparameters we tuned can be found in Appendix. For our k-nearest neighbor classifier, we also choose $k = N$, where N is the number of traces per class target training

[1]https://www.alexa.com/topsites

Table 1: Attack accuracy (%) of TF (Source: AWF775, Target: Wang100; $M = 25, T = 70$). The results are in the form of mean ± standard deviation.

Method	$N = 1$	$N = 5$	$N = 10$	$N = 15$	$N = 20$
TF (obtained in this paper)	71.3 ± 2.0	84.2 ± 0.5	86.1 ± 0.4	86.7 ± 0.5	87.0 ± 0.5
TF (reported in CCS19 [36])	73.1 ± 1.8	84.5 ± 0.4	86.2 ± 0.4	86.6 ± 0.3	87.0 ± 0.3

Table 2: Attack accuracy (%) of Fine-tuning, TF and Ours ($M = 25, T = 70$).

Source	Target	Method	$N = 1$	$N = 5$	$N = 10$	$N = 15$	$N = 20$
AWF100	Wang100	Fine-tuning	46.1 ± 2.68	68.2 ± 0.67	76.5 ± 1.39	80.4 ± 1.51	82.0 ± 1.22
		TF	**55.5 ± 1.76**	**72.7 ± 0.78**	75.8 ± 0.45	76.4 ± 0.56	77.3 ± 0.37
		Ours (AF-SingleSource)	30.4 ± 3.46	64.0 ± 2.03	**82.1 ± 1.86**	**87.7 ± 1.30**	**89.3 ± 1.30**
AWF100	DF95	Fine-tuning	37.7 ± 2.46	**57.7 ± 1.53**	67.8 ± 1.34	73.0 ± 6.81	79.6 ± 1.33
		TF	**38.3 ± 2.11**	55.4 ± 1.11	59.9 ± 1.07	61.4 ± 0.76	62.6 ± 0.60
		Ours (AF-SingleSource)	28.0 ± 1.71	53.7 ± 3.05	**73.4 ± 1.72**	**80.3 ± 1.27**	**82.4 ± 0.91**
AWF100	AF100	Fine-tuning	26.3 ± 2.23	33.7 ± 1.79	46.4 ± 0.95	52.0 ± 2.85	50.7 ± 1.51
		TF	30.0 ± 1.18	37.9 ± 0.78	39.3 ± 0.65	40.4 ± 0.52	40.5 ± 0.56
		Ours (AF-SingleSource)	**31.0 ± 2.88**	**49.0 ± 2.65**	**60.7 ± 1.06**	**65.5 ± 1.77**	**67.9 ± 1.29**

data. We also use the Mean Embedded Vector of N samples to train k-nearest neighbor in the closed-world evaluation to have a fair comparison with Triplet Fingerprinting.

5.3 Closed-World Evaluation

Experiment A.1: Reproducing Results of Triplet Fingerprinting. We reproduce the results of Triplet Fingerprinting [36] in the closed-world evaluation. We use the same datasets and same parameters as the ones in Triplet Fingerprinting. We choose 1 source dataset and 1 target dataset as follows:

- Source: AWF775; Target: Wang100

We use M to denote the number of traces per class used in pre-training, N as the number of traces per class in target training data and T as the number of traces per class in target test data. By following the same approach in [36], we randomly select $M = 25$ traces per class from the source dataset in pre-training. For the target dataset, we take 90 traces in each class, where these 90 traces are divided into 2 chunks. The 20 traces in thefi rst chunk can be used for training and the 70 traces in the second chunk are used for testing. A number of N traces out of the 20 traces in thefirst chunk are used to train the k-nearest neighbor classifier, where $N = \{1, 5, 10, 15, 20\}$. Unless specified, we choose the same values for M, N and T in other experiments. As shown in Table 1, we obtained almost the same results as the ones reported in [36].

Experiment A.2: Comparison among Fine-tuning, Triplet Fingerprinting and Our Method. We compare the attack performance of three methods, includingfi ne-tuning, Triplet Fingerprinting, and Adaptive Fingerprinting. We evaluate the case where 1 source dataset and 1 target dataset are selected. To be more consistent with our later experiments, which investigate impacts of several parameters/aspects, we select the source dataset and target data with the same or a similar number of classes.

Specifically, we choose AWF100 (rather than AWF775 in Experiment A.1) as the source dataset and we pick Wang100, DF95, or AF100 as the target dataset respectively as below:

- Source: AWF100; Target: Wang100
- Source: AWF100; Target: DF95
- Source: AWF100; Target: Our100

Attack Accuracy. As shown in Table 2, Triplet Fingerprinting is the most effective method when the number of training samples from the target dataset is 1. However, when $N \geq 10$, our method AF-SingleSource achieves the highest accuracy. This observation is consistent even when we select a different target dataset. In some cases (e.g., source is AWF100 and target is AF100), our method can also outperform the other two methods when $N = 5$.

When N is extremely small (e.g., 1 or 5), our method does not have sufficient target traces involved to pre-train a good Domain Discriminator. On the other hand, the other two methods do not need target data in pre-training. This difference is likely the reason that our method is often less effective when $N = \{1, 5\}$. When AF100 is the target, the performance of all the three methods are less effective compared to other target datasets. This is likely because AF100, which is collected more recently with a much different version of Tor Browser, is less similar to the source AWF100 among the three target datasets that we examined.

Note that our results offi ne-tuning are higher than the ones reported in [36]. As we use the same source code, the DF model, and parameters as in [36], we believe the potential reason is that we use AWF100 rather than AWF775 as the source data, where the number of classes in a source target is the same or similar as the number of classes in a target dataset. This likely benefitsfine-tuning for obtaining higher accuracy.

Feature Visualization. We also visualize the feature spaces obtained from the three methods by using t-SNE (t-Distributed Stochastic Neighbor Embedding) [25], which is a method visualizing high-dimensional data into a 2-dimensional space. Specifically, we extract the last layer of the DF model trained in each of these 3 methods, where AWF100 serves as the source and Wang100 serves as the target. As we can observe in Fig. 10, all the three methods obtain a feature space that can distinguish data better from different classes compared to the space of target data.

Pre-Training Time. We also compare the pre-training time among the three methods. As shown in Table 3, Fine-tuning is the most efficient in pre-training time. Triplet Fingerprinting requires much longer time than the other two as mining triplets is a key but time-consuming step in its pre-training. The pre-training time offi ne-tuning and Triplet Fingerprinting do not depend on

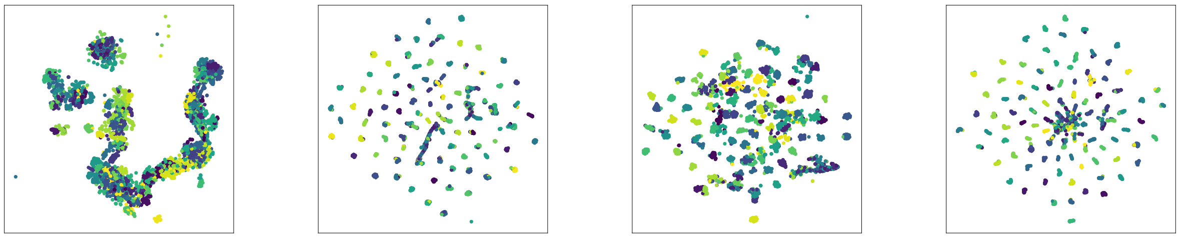

(a) Target Data (b) Fine-tuning (c) Triplet Fingerprinting (d) Adaptive Fingerprinting (Ours)

Figure 10: The visualization of feature spaces using t-SNE (Source: AWF100; Target: Wang100; $M = 25$ and $N = 20$).

Table 3: Pre-training Time (seconds) of Fine-tuning, TF and Ours ($M = 25, T = 70$).

Source	Target	Method	$N = 1$	$N = 5$	$N = 10$	$N = 15$	$N = 20$
AWF100	Wang100	Fine-tuning	27				
		TF	772				
		Ours (AF-SingleSource)	177	180	183	183	182

Table 4: The impact of different source datasets on attack accuracy (%), where $M = 25$ and $T = 70$.

Source	Target	Method	$N = 1$	$N = 5$	$N = 10$	$N = 15$	$N = 20$
AWF100	Wang100	Fine-tuning	46.1	68.2	76.5	80.4	82.0
		TF	**55.5**	**72.7**	75.8	76.4	77.3
		Ours (AF-SingleSource)	30.4	64.0	**82.1**	**87.7**	**89.3**
DF95	Wang100	Fine-tuning	45.8	66.9	78.3	82.3	83.6
		TF	**55.2**	**72.3**	75.3	75.9	76.4
		Ours (AF-SingleSource)	32.5	71.1	**83.4**	**87.6**	**88.7**
AF100	Wang100	Fine-tuning	**46.5**	69.0	79.9	84.0	84.4
		TF	41.3	57.4	61.3	62.8	63.5
		Ours (AF-SingleSource)	28.3	**70.9**	**83.5**	**87.3**	**88.8**

the selection of the target data or the number of traces N per class in the target training data. Our method, AF-SingleSource, has a slightly different pre-training time given a different N as the target training data are involved in the inputs of a domain adversarial network. When we change the target dataset to DF95 or AF100, we have the same pre-training time forfi ne-tuning and TF and almost the same pre-training time for our method AF-SingleSource. We skip further details due to space limitation.

Experiment A.3: Impact of Different Source Datasets. We evaluate the impact on the attack accuracy of the three methods if an attacker uses a different source dataset given the same target dataset. Specifically, we still examine the cases of 1 source dataset and 1 target dataset with the following

- Source: AWF100; Target Wang100
- Source: DF95; Target: Wang100
- Source: AF100; Target: Wang100

As shown in Table 4, with different source datasets, our method AF-SingleSource is always the most effective one among the three methods when $N \geq 10$. In addition, when $N = 5$, our method can also outperform other methods in some cases (e.g., when the source dataset is AF100). Triplet Fingerprinting is always more effective than our method when $N = 1$. Bothfi ne-tuning and our method achieves a similar attack accuracy regardless which source dataset is utilized. On the other hand, we notice that the attack accuracy of Triplet Fingerprinting varies significantly when a different source dataset is selected given the same target dataset.

Experiment A.4: Comparisons between AF-SingleSource and AF-MultiSource. We compare the two versions of our method AF-SingleSource and AF-MultiSource. In other words, we investigate the impact of the number of source datasets on attack accuracy. Specifically, for AF-SingleSource, it takes one source dataset and target training data in the pre-training. For AF-MultiSource, it takes multiple source datasets in the pre-training. The training and testing steps are the same between the two versions. The numbers of traces per class in source data, target training data and target test data (i.e., $M = 25$, $N = \{1, 5, 10, 15, 20\}$ and $T = 70$) remain the same as the ones in the previous experiments.

As shown in Table 5, given the target dataset is Wang100, when AF-MultiSource takes multiple source datasets, such as 2 source datasets or 3 source datasets, but no target dataset in the pre-training phase, it can achieve a greater accuracy than AF-SingleSource, especially when N is small, such as 1 and 5. This is expected as when N is much smaller than M, the inputs in a domain adversarial network used in AF-SingleSource are unbalanced between the source and target. As a result, AF-SingleSource is not able to learn a better domain-invariant feature space. On the other hand, AF-MultiSource does not rely on target data to obtain a domain-invariant feature space in the pre-training.

Table 5: The impact of the number of source datasets in our method on attack accuracy (%), where $M = 25$ and $T = 70$.

Method	Pre-Training Setting	Source	Target	$N = 1$	$N = 5$	$N = 10$	$N = 15$	$N = 20$
AF-SingleSource	1 Source 1 Target	AWF100	Wang100	30.4	64.0	82.1	87.7	89.3
AF-MultiSource	2 Source 0 Target	{AWF100, DF95}	Wang100	**37.6**	**72.4**	84.4	86.8	**89.8**
AF-MultiSource	3 Source 0 Target	{AWF100, DF95, AF100}	Wang100	33.8	72.2	**85.8**	**88.5**	88.1

Table 6: The impact of the number of traces per class in a source dataset (Source: AWF100; Target: Wang100), where $N = 20$ and $T = 70$.

Metric	Method	$M = 25$	$M = 50$	$M = 100$	$M = 200$	$M = 400$
Accuracy (%)	Fine-tuning	82.0 ± 1.22	84.0 ± 0.93	85.7 ± 0.58	87.5 ± 0.69	87.8 ± 0.99
	TF	77.3 ± 0.37	79.5 ± 0.33	78.7 ± 0.25	78.3 ± 0.55	78.5 ± 0.44
	Ours (AF-SingleSource)	**89.1 ± 0.77**	**88.4 ± 0.40**	**89.0 ± 0.78**	**88.8 ± 0.90**	**88.9 ± 0.49**
Pre-Training Time (second)	Fine-tuning	27	45	70	136	249
	TF	**772**	**2,913**	**11,761**	**38,163**	**233,999**
	Ours (AF-SingleSource)	182	182	183	188	189

Table 7: Attack accuracy (%) of Fine-tuning, TF and Ours (Source: DF95-Defended, Target: Wang100-Defended, $M = 25$, $T = 70$) in the closed-world setting.

Method	$N = 1$	$N = 5$	$N = 10$	$N = 15$	$N = 20$
Fine-tuning	**19.1 ± 1.6**	33.9 ± 1.3	47.3 ± 1.6	53.6 ± 1.4	56.8 ± 1.9
TF (based on results reported in CCS19 [36])	15.5 ± 1.7	**39.8 ± 0.5**	47.2 ± 1.1	50.1 ± 0.4	51.7 ± 0.5
Ours (AF-SingleSource)	9.13 ± 1.0	37.3 ± 1.8	**56.9 ± 2.4**	**65.7 ± 1.2**	**70.2 ± 0.9**

Experiment A.5: Impact of the Number of Traces per Class in a Source dataset. We examine if the number of traces per class in a source dataset increases, what impacts it may have on website fingerprinting. We still compare the three methods, includingfine-tuning, Triplet Fingerprinting, and our method. We take 1 dataset as source and 1 dataset as the target.

- Source: AWF100; Target: Wang100

where we choose the number of traces per class in a source dataset as $M = \{25, 50, 100, 200, 400\}$. For the target dataset, we choose $N = 20$ and $T = 70$.

As shown in Table 6, bothfi ne-tuning and our method outperform Triplet Fingerprinting when the number of traces per class M is greater than or equal to 25. This is consistent with our observations in previous experiments. In addition, both our method and Triplet Fingerprinting remain at the same level of attack accuracy when the value of M increases. The accuracy offi ne-tuning slightly increases with an increase on M. When $M = 400$, the accuracy of fine-tuning gets very close to the accuracy of our method but it requires longer pre-training time.

Moreover, we notice that bothfi ne-tuning and our method are very efficient in pre-training time while Triplet Fingerprinting requires significant amounts of pre-training time as the value of M increases. For instance, when $M = 400$, Triplet Fingerprinting requires more than 64 hours in pre-training while our method only takes 3 minutes. This is because when M increases, Triplet Fingerprinting needs to evaluate a much greater number of triplets in pre-training, which is time-consuming. In other words, our method is more scalable than Triplet Fingerprinting when there are more data from a source dataset.

Experiment A.6: Impact of Defenses Against Website Fingerprinting. In this experiment, we investigate if traffic traces are protected by existing defenses, what impact it may have. Specifically, we choose WTF-PAD [20] as the defense method to obfuscate traffic traces. We take 1 dataset as source and 1 dataset as the target. As generating defended data with WTF-PAD needs timestamps of packets, which are not available in some of the datasets we use. We choose the source and target as below

- Source: DF95 (Defended with WTF-PAD); Target: Wang100 (Defended with WTF-PAD)

We choose $M = 25$, $N = \{1, 5, 10, 15, 20\}$, and $T = 70$. We use the default parameters and the source code of WTF-PAD from [1] to generate defended data.

As shown in Table 7, we have similar observations as previous experiments, where our method performs better than the other two when $N \geq 10$. On the other hand, the accuracy of all the three methods over defended data decreases compared to the results over non-defended data in Table 5.

5.4 Open-World Evaluation

Experiment B.1: Reproducing Results of Triplet Fingerprinting in the Open-World Setting. In this experiment, we reproduce the results of Triplet Fingerprinting in the open-world evaluation. We follow the same datasets as the ones in [36]. Specifically, we choose the source and the target as follows:

- Source: AWF775; Target: {Wang100, AWF9000}

where Wang100 is used as data for monitored websites of the target dataset and AWF9000 is used as data for unmonitored websites of the target dataset. Each unmonitored website only has 1 trace. Only monitored websites of a source dataset are utilized in pre-training.

For the source dataset, we still choose $M = 25$ traces per website in pre-training. For the monitored websites in the target dataset, $N = \{5, 10, 15, 20\}$ traces per monitored website are examined in training and $T = 70$ traces per monitored website are used in testing. However, how many traces of unmonitored websites used in training and testing were not specifically described in [36]. To reproduce the results, we explore the following two scenarios:

Table 8: Precision and Recall of TF (Source AWF775, Target: {Wang100, AWF9000}; $M = 25$) in the open-world setting.

Tuned for Precision

Method	$N = 5$		$N = 10$		$N = 15$		$N = 20$	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
TF (Scenario 1, obtained in this paper)	0.633	0.901	0.644	0.876	0.690	0.871	0.682	0.871
TF (Scenario 2, obtained in this paper)	0.954	0.870	0.961	0.872	0.969	0.852	0.970	0.855
TF (results reported in [36])	0.973	0.831	0.953	0.879	0.944	0.903	0.933	0.907

Tuned for Recall

Method	$N = 5$		$N = 10$		$N = 15$		$N = 20$	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
TF (Scenario 1, obtained in this paper)	0.457	0.999	0.453	0.999	0.455	0.999	0.457	0.999
TF (Scenario 2, obtained in this paper)	0.921	0.966	0.936	0.970	0.943	0.970	0.945	0.972
TF (results reported in [36])	0.950	0.950	0.922	0.971	0.908	0.983	0.905	0.978

Table 9: Precision and recall of Fine-tuning, TF, and Ours ($M = 25$, $N = 10$, $T = 70$, $N' = 1000$, $T' = 7000$).

Source	Target	Method	Tuned for Precision		Tuned for Recall	
			Precision	Recall	Precision	Recall
AWF100	{Wang100, Wang9000}	Fine-tuning	0.999	0.871	0.999	0.937
		TF	0.937	0.862	0.921	0.943
		Ours (AF-SingleSource)	0.998	0.729	0.996	0.997
AWF100	{DF95, DF9000}	Fine-tuning	0.990	0.596	0.981	0.680
		TF	0.864	0.570	0.834	0.652
		Ours (AF-SingleSource)	0.997	0.562	0.993	0.995
AWF100	{AF100, AF9000}	Fine-tuning	0.986	0.215	0.975	0.259
		TF	0.816	0.469	0.780	0.613
		Ours (AF-SingleSource)	0.998	0.461	0.996	0.994

- ***Scenario 1: Same Number*** ***($N' = N$).*** We divide all the 9,000 traces in AWF9000 into 2 chunks, where thefi rst chunk includes 20 traces and the second chunk includes 8,980 traces. Then, we choose $N' = \{5, 10, 15, 20\}$ traces randomly from thefi rst chunk in training and $T' = 8,980$ traces in testing.
- ***Scenario 2: Same Ratio*** ***($N/T = N'/T'$).*** We divide all the 9,000 traces in AWF9000 into 2 chunks, where thefi rst chunk includes 2,000 traces and the second chunk includes 7,000 traces. Then, we choose $N' = \{500, 1000, 1500, 2000\}$ traces randomly from thefi rst chunk in training and $T' = 7,000$ traces in testing.

As illustrated in Table 8, the results from the second scenario are more similar to the results reported in [36]. Therefore, we use the second scenario to select training and test data in our later experiments in the open-world evaluation. In fact, when using the second scenario, the data from monitored websites and the data from unmonitored websites are more balanced. It is likely why the second scenario derives better results. Note that when we re-train the k-nearest neighbor classifier for Triplet Fingerprinting (as well as our method examined later) in the open-world setting, we directly use traces rather than the Mean Embedded Vector of them. This is because that using Mean Embedded Vectors derives a much lower performance in our open-world evaluation.

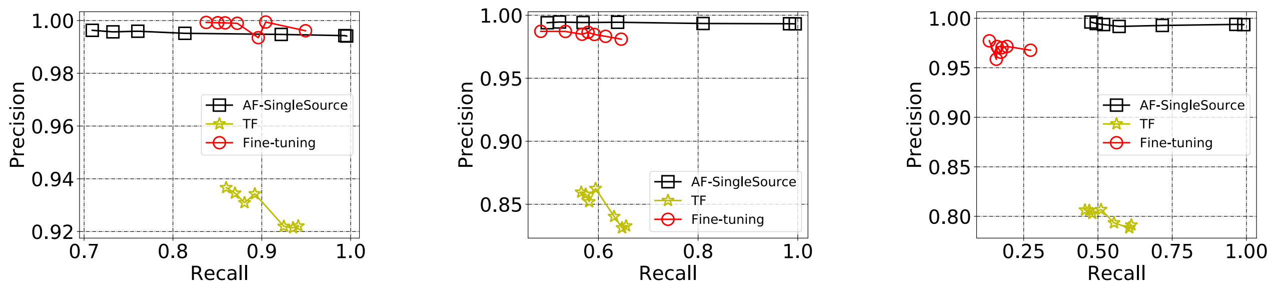
When we tune the confidence threshold for precision or recall, we use the threshold range $[0, 0.4]$ in this paper. We also explore thresholds that are greater than 0.4 in our experiments. However, it can easily end up with either an extremely high precision with a low recall or a low precision with an extremely high recall. Note that even with the second scenario, the results we obtained in Table 8 are not exactly the same as the ones reported in [36]. This is likely because thefi rst chunk of traces from unmonitored websites was randomly selected and the thresholds tuned for precision and recall might be different in [36].

Scenario 1 does not derive promising results as Scenario 2. This is because data from unmonitored websites consist of 1 sample per website, which introduces high variance over samples across 9,000 unmonitored websites. Taking $N = \{5, 10, 15, 20\}$ samples in training while using $T = 8,980$ samples in testing in Scenario 1 would be challenging to derive both high precision and high recall. The high variance over samples across unmonitored websites could also be the reason that training k-nearest neighbor classifier with Mean Embedded Vectors leads to a lower performance.

Experiment B.2: Comparison among Fine-tuning, Triplet Fingerprinting and Our method. In this experiment, we compare the attack performance of the three methods in the open-world evaluation. We evaluate the case where 1 source dataset and 1 target dataset are selected. To be consistent with our experiments in the closed-world setting, we select the the datasets as follows:

- Source: AWF100; Target: {Wang100, Wang9000}
- Source: AWF100; Target: {DF95, DF9000}
- Source: AWF100; Target: {AF100, AF9000}

where Wang100, DF95 and AF100 serve as monitored websites respectively and Wang9000, DF9000 and AF9000 serve as unmonitored websites respectively. We use the same parameters as the ones in the last experiment. We summarize our results in Table 9 and also in Fig. 11, where we choose $M = 25$, $N = 10$, $T = 70$, $N' = 1,000$ and $T' = 7,000$. As we can observe from the precision-recall curves in Fig. 11, in general, our method derives a better results in the open-world setting compared to the other two methods.



(a) Source: AWF100; Target: {Wang100, Wang9000} (b) Source: AWF100; Target: {DF95, DF9000} (c) Source: AWF100; Target: {AF100, AF9000}

Figure 11: Precision-recall curve of the three methods in the open-world evaluation ($M = 25, N = 10, T = 70, N' = 1000, T' = 7000$).

6 RELATED WORKS

Most of the studies formulate websitefi ngerprinting as a supervised learning problem and address it with machine learning. For instance, studies such as k-NN [43], CUMUL [29] and k-FP [13], manually select features and utilize traditional machine learning algorithms. Recent studies [4, 5, 27, 31, 35] leveraged deep neural networks to automatically extract features and can obtain higher accuracy than the ones that rely on traditional machine learning algorithms. For example, Sirinam et al [35] proposed Deep Fingerprinting, which is built upon Convolutional Neural Networks and achieves 98% accuracy in the closed-world setting over 95 websites with 1,000 traces per site. However, to achieve a higher accuracy, an attack often needs to collect large-scale traffic dataset, which is time-consuming. Sirinam et. al. [36]fi rst proposed to use transfer learning to perform websitefi ngerprinting over few encrypted traffic to address the need of recollecting large-scale datasets.

Besides the challenges of relying on large-scale training data, other challenges in websitefi ngerprinting have also been examined. For instance, studies in [8, 46] addressed websitefi ngerprinting in the multi-tab scenario, where a user could open multiple websites at the same time and the traffic of these websites are overlapped. Juarez et. al. [19] discussed the base rate fallacy in the open-world evaluation. Li et. al. [22] suggested that leveraging entropy can measure the privacy leakage more accurately rather than using accuracy. Wang [41] proposed to use r-precision to measure the performance of websitefi ngerprinting in the open-world evaluation. Shusterman et. al. [34] leveraged cache pattern of a user's computer rather network traffic to infer which website a user visits.

Defenses [6, 7, 9, 11, 14, 17, 20, 26, 45] against websitefingerprinting have also been studied. The primary approach is to obfuscate traffic pattern such that the traces of different websites are similar, which is more difficult to distinguish for an attacker. Wang et al. [45] proposed a defense, named Walkie-Talkie, which combines traffic traces of two different websites into a super sequence. Gong and Wang [11] proposed two defenses, FRONT and GLUE. FRONT introduces higher perturbations to early packets in a traffic to hide more important features against websitefi ngerprinting. GLUE adds dummy traffic between two traffic traces such that it is more difficult for an attacker to identify the beginning of each traffic trace. Two studies [17, 26] also leverage adversarial examples of encrypted traffic to mitigate the attack accuracy against deep-learning-based attacks. Cadena et. al. proposed TrafficSilver [6], which modifies traffic pattern by splitting traffic into multiple Tor relays.

7 DISCUSSIONS AND FUTURE WORK

Data Augmentation. Data Augmentation [16, 33], which can create new samples for labeled data, is often used with transfer learning to boost performance in classifications. We did not examine data augmentation in our study. This is because existing data augmentation methods, which work well for images, cannot be directly used for encrypted traffic.

For instance,fl ipping (from left to right) or rotating (with a certain angle) an image can output a new image. However, a traffic trace cannot be rotated as there are no angles. Flipping the packets from the very end to the beginning of a traffic trace does not derive a valid traffic trace that can happen in the real world. We will leave whether it is feasible to perform data augmentation over encrypted traffic (or network traffic in general) for future work.

Other Machine Learning Models. Some other models, such as deep forest [48], need minimal efforts on tuning hyper-parameters and can also achieve promising results with small-scale training data in the image domain. Whether it can derive promising results in the context of websitefi ngerprinting remains unknown and we will leave it for future work.

8 CONCLUSION

We propose Adaptive Fingerprinting, which can perform website fingerprinting over few encrypted traffic. Our experimental results show that our method can derive high accuracy over a new but small-scale dataset by transferring knowledge learned from a previous large-scale dataset. Our method can outperform previous methods over few encrypted traffic (except when only 1 trace is available to each monitored website). Moreover, our method is more efficient in terms of pre-training time compared to previous studies. Our results suggest that, by leveraging transfer learning, the bootstrap time of websitefi ngerprinting can be reduced and website fingerprinting can be more practical in the real world.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers and our shepherd, Dr. Phani Vadrevu, for their insightful comments and suggestions. The authors also would like to thank Dr. Marc

Juarez and Vera Rimmer for explaining details regrading Tor traffic collection, thank Dr. Matthew Wright for providing help when reproducing the results of Triplet Fingerprinting, and thank Ohio Cyber Range at UC for providing multiple virtual machines to facilitate Tor traffic collection. The authors were partially supported by National Science Foundation (CNS-1947913) and UC Office of the Vice President for Research - Pilot Program.

REFERENCES

[1] 2016. WTF-PAD. https://github.com/wtfpad/wtfpad

[2] 2018. tor-browser-crawler. https://github.com/onionpop/tor-browser-crawler

[3] 2021. AdaptiveFingerprinting. https://github.com/SmartHomePrivacyProject/AdaptiveFingerprinting

[4] K. Abe and S. Goto. 2016. Fingerprinting Attack on Tor Anonymity Using Deep Learning. In *Proc. of Aisa Pacific Advanced Network (APAN).*

[5] S. Bhat, D. Lu, A. Kwon, and S. Devadas. 2019. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. In *Proc. of PETS'19.*

[6] W. D. Cadena, A. Mitseva, J. Hiller, J. Penekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko. 2020. TrafficSliver: Finghting Website Fingerprinting Attacks with Traffic Splitting. In *Proc. of ACM CCS'20.*

[7] X. Cai, R. Nithyanand, and R. Johnson. 2014. CS-BuFLO: A Congrestion Sensitive Website Fingerprinting Defense. In *Proc. of 13th ACM Workshop on Privacy in Electronic Society.*

[8] W. Cui, T. Chen, C. Fields, J. Chen, A. Sierra, and E. Chan-Tin. 2019. Revisting Assumtions for Website Fingerprinting Attacks. In *Proc. of ACM ASIACCS'19.*

[9] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proc. of IEEE S&P'12.*

[10] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. 2016. Domain-Adversarial Tranining of Neural Networks. *Journal of Machine Learning Research* (2016).

[11] Jiajun Gong and Tao Wang. 2020. Zero-delay Lightweight Defenses against Website Fingerprinting. In *Proc. of USENIX Security'20.*

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. Generative Adversarial Nets. In *Proc. of the International Conference on Nerual Information Processing Systems (NIPS 2014).*

[13] J. Hayes and G. Danezis. 2016. K-Fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *Proc. of USENIX Security'16.*

[14] S. Henri, G. Garcia-Aviles, P. Serrano, A. Banchs, and P. Thiran. 2020. Protecting against Website Fingerprinting with Multihoming. In *Proc. of PETS'20.*

[15] Dominik Hermann, Rolf Wendolsky, and Hannes Federrath. 2009. Website Fingertinging: Attacking Popular Privacy Enhancing Tehnologies with the Multinomial Naive-Bayes Classifier. In *Proc. of ACM Workshop on Cloud Computing Security.*

[16] D. Ho, E. Liang, I. Stoica, P. Abbeel, and X. Chen. 2019. Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules. In *Proc. of ICML'19.*

[17] M. Imani, M. S. Rahman, N. Mathews, and M. Wright. 2019. Mockingbird: Defending Against Deep-Learning-Based Website Fingerprinting Attacks with Adversarial Traces. (2019). https://arxiv.org/pdf/1902.06626.pdf.

[18] R. Jansen, M. Juarez, R. Galvez, T. Elahi, and C. Diaz. 2018. Inside Job: Applying Traffic Analysis to Measure Tor from Within. In *Proc. of NDSS'18.*

[19] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. 2014. A Criticial Evaluation of Website Fingerprinting Attacks. In *Proc. of ACM CCS'14.*

[20] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright. 2016. Toward an Efficient Website Fingerprinting Defense. In *Proc. of ESORICS'16.*

[21] G. Koch, R. Zemel, and R. Salakhutdinov. 2015. Siamese Neural Networks for One-shot Image Recognition. In *Proc. of the 32th International Conference on Machine Learning (ICML'15).*

[22] Shuai Li, Huajun Guo, and Nicholas Hopper. 2018. Measuring Information Leakage in Website Fingerprinting Attacks and Defenses. In *Proc. of ACM CCS'18.*

[23] Marc Liberatore and Brian Neil Levine. 2006. Inferring the Source of Encrypted HTTP Connections. In *Proc. of ACM CCS'06.*

[24] M. Long and J. Wang. 2015. Learning transferable features with deep adaptation networks. In *Proc. of ICML'15.*

[25] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[26] M. Nasr, A. Bahramali, and A. Houmansadr. 2020. Blind Adversarial Network Perturbations. (2020). https://arxiv.org/pdf/2002.06495.pdf.

[27] Se Eun Oh, S. Sunkam, and N. Hopper. 2019. p-FP: Extraction, Classification, and Predication of Website Fingerprints. In *Proc. of PETS'19.*

[28] S. J. Pan and Q. Yang. 2009. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* (2009).

[29] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Penekamp, K. Wehrle, and T. Engel. 2016. Website Fingerprinting at Internet Scale. In *Proc. of NDSS'16.*

[30] O. M. Parki, A. Vedaldi, and A. Zisserman. 2015. Deep Face Recognition. In *British Machine Vision Association.*

[31] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *Proc. of NDSS'18.*

[32] F. Schroff, D. Kalenichenko, and J. Philbin. 2015. FaceNet: A Unified Embedding for Face Recognition and Clustering. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[33] C. Shorten and T. M. Khoshgoftaar. 2019. A Survey on Image Data Augmentation for Deep Learning. *Journal of Big Data* 6, 60 (2019).

[34] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom. 2019. Robust Website Fingerprinting Through the Cache Occupancy Channel. In *Proc. of USENIX Security'19.*

[35] P. Sirinam, M. Imani, M. Juarez, and M. Wright. 2018. Deep Fingerprinting: Understanding Website Fingerprinting Defenses with Deep Learning. In *Proc. of ACM CCS'18.*

[36] 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.

[37] B. Sun, J. Feng, and K. Saenko. 2016. Return of Frustratingly Easy Domain Adaption. In *Proc. of AAAI Conference on Artificial Intelligence.*

[38] Y. Taigman, M. Yang, M. A. Ranzato, and L. Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proc. of IEEE CVPR'14.*

[39] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. 2017. Adversarial Discriminative Domain Adaptation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[40] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. 2014. Deep domain confusion: Maximizing for domain invariance. (2014). https://arxiv.org/pdf/1412.3474.pdf.

[41] Tao Wang. 2020. High Precision Open-World Website Fingerprinting. In *Proc. of IEEE S&P'20.*

[42] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for websitefi ngerprinting. In *23rd {USENIX} Security Symposium ({USENIX} Security 14).* 143–157.

[43] Tao Wang, Xiang Cui, Rishab Nithyannand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks on Provable Denfenses for Website Fingerprinting. In *Proc. of 23rd USENIX Security Symposium.*

[44] T. Wang and I. Goldberg. 2016. On Realistically Attacking Tor with Website Fingerprinting. In *Proc. of PETS'16.*

[45] T. Wang and I. Goldberg. 2017. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *Proc. of USENIX Security'17.*

[46] Y. Xu, T. Wang, Q. Li, Q. Gong, Y. Chen, and Y. Jiang. 2018. A Multi-Tab Website Fingerprinting Attack. In *Proc. of ACSAC'18.*

[47] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems.* 3320–3328.

[48] Zhi-Hua Zhou and Ji Feng. 2017. Deep Forest: Towards an Alternative to Deep Neural Networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence.* 3553–3559.

[49] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *Proc. IEEE* (2020).

A APPENDIX

Tuned Hyperparameters. The tuned hyperparameters of Domain Discriminator and Source Classifier in our method (AF-SingleSource) are described in Table 10. We run grid search and select the ones that derive the highest accuracy in the closed-world evaluation. In our experiments, we set tradeoffparameter λ (in Eq. 4) as 1 and the value of learning rate α (in Eq. 4) as 1×10^{-5}.

Table 10: Tuned Hyperparameters in AF-SingleSource.

Parameters	Search Space	Tuned
Classifier Layer Type	Convolution, Fully-connected	Convolution
Classifier Depth	$\{2, 3, 4\}$	2
Discriminator Layer Type	Convolution, Fully-connected	Convolution
Discriminator Depth	$\{2, 3, 4\}$	3
Embedded Vector Size	$\{64, 128, 256, 512\}$	512