Adaptive Encrypted Traffic Fingerprinting With Bi-Directional Dependence

Khaled Al-Naami, Swarup Chandra, Ahmad Mustafa, Latifur Khan, Zhiqiang Lin,
Kevin Hamlen, and Bhavani Thuraisingham
Computer Science Department
The University of Texas at Dallas
Richardson, TX, 75080
{khaled.al-naami, swarup.chandra, ahmad.mustafa, lkhan, zhiqiang.lin,
hamlen, bhavani.thuraisingham}@utdallas.edu

Abstract

Recently, network traffic analysis has been increasingly used in various applications including security, targeted advertisements, and network management. However, data encryption performed on network traffic poses a challenge to these analysis techniques. In this paper, we present a novel method to extract characteristics from encrypted traffic by utilizing data dependencies that occur over sequential transmissions of network packets. Furthermore, we explore the temporal nature of encrypted traffic and introduce an adaptive model that considers changes in data content over time. We evaluate our analysis on two packet encrypted applications: website fingerprinting and mobile application (app) fingerprinting. Our evaluation shows how the proposed approach outperforms previous works especially in the open-world scenario and when defense mechanisms are considered.

1. INTRODUCTION

With a tremendous growth in the number of Internet users over the past decade, network *traffic analysis* has gained significant interest in both academia and industry. Applications such as personalized marketing [22] and traffic engineering [30,31] have spurred the demand for tracking online activities of users [24]. For example, by tracking the websites accessed by a particular user, related products may be advertised. Unfortunately, online users have fallen victim to adversaries who use such tracking mechanisms for malicious activities by passively monitoring network traffic. As a result, encryption technologies such as SSL/TLS are used extensively to hide data in network traffic from unauthorized access. In addition to data encryption, end-node network identifiers (e.g. IP addresses) may also be hidden from external adversaries using technologies such as Tor [14], to anonymize the user.

Recent studies [7,36] on traffic analysis have focused on identifying characteristic patterns in network traffic that reveal the behavior of an end-node, thereby de-anonymizing the network. Essentially, pattern recognition techniques are employed over features extracted from encrypted network traffic passively captured at the user's end. This behavior identification process of an end-node (i.e. either a service accessed by the user, or an application at the user's end involved in the network traffic) is called *Traffic Fingerprinting*.

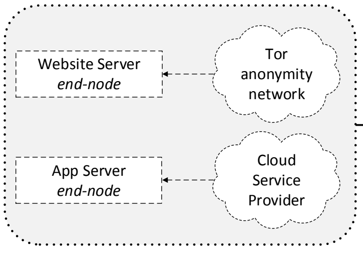

Figure 1: Illustration of website and app fingerprinting

In this paper, we focus on the following two applications (illustrated in Figure 1) whose primary goal is to perform traffic fingerprinting to identify an end-node generating encrypted traffic. Here, a man-in-the-middle (i.e., network administrator, ISP, government agency, etc) captures encrypted network traffic passively at the user's end.

Website Fingerprinting. This application involves identifying the webpage (end-node) accessed by a user who actively hides online activities using an anonymity network such as Tor. Knowledge of the user's online activities may be useful in applications such as targeted advertisements, tracking terrorist activities, checking for DRM violations, etc. On the contrary, it violates the user's online privacy. Destination IP addresses obtained from encrypted traffic in this setting cannot be used for webpage identification since they would be encapsulated by the encryption scheme. Fingerprinting over such encrypted data for identification of webpage (or website) is widely known as *Website Fingerprinting* [25]. We denote this as WFIN.

App Fingerprinting. Unlike websites, smartphone apps access the Internet by connecting to remote services that provide necessary data for their operation. Examples of such services include advertisements, 3rd party libraries, and other API-based services. Applications, such as ad relevance, network bandwidth management, and app recommendations, may require the knowledge of apps running on a particular device in order to improve user experience. On the other hand, an adversarial view of such knowledge may lead to initiation of targeted attacks [39] involving known vulnerabilities in apps. While apps do not hide the destination IP addresses, they may access multiple overlapping destinations. For example, two apps may access the same 3rd-party library while utilizing the service in a distinct manner. For a man-in-the-middle observing network traffic, identifying the two apps on the same device is hard when relying only on the IP addresses. However, the apps may have distinct network traffic patterns useful for discrimination. We call the identification of apps on a device, using their encrypted network traffic patterns, *App Fingerprinting*, denoted by AFIN.

ACSAC '16, December 05-09, 2016, Los Angeles, CA, USA
© 2016 ACM. ISBN 978-1-4503-4771-6/16/12. . . $15.00
DOI: http://dx.doi.org/10.1145/2991079.2991123

A fundamental challenge in performing traffic fingerprinting over encrypted data is the identification of characteristic features, which are often used in machine learning classifiers. In particular, encrypted traffic consists of network packets that carry application data along with other control messages depending on the communication protocol. In general, a protocol such as TCP limits the size of each packet. Moreover, each packet incurs a finite transmission time depending on the network path followed from its source to its destination. When a man-in-the-middle passively captures a sequence of packets flowing at the user's end, the packet size, time-stamp, and direction can be observed to form a set of features. As the goal of fingerprinting is to determine end-node patterns, one must consider a sequence of network packets in the captured traffic generated during a communication session involving the end-node under investigation. We call this sequence of packets a *trace*.

Over time, the captured network traffic may contain multiple traces associated with a set of end-nodes with different sessions initiated by the same user. In this setting, feature extraction is performed over each trace by combining features of each of its packets in a suitable manner [6,7,21,25,27,36]. Most existing techniques combine features by assuming independence between subsequent transmissions [6,27]. Therefore, relationship between packets in a TCP session, occurring consecutively in opposite directions (*viz.*, uplinks from user to server, or downlinks from server to user), are ignored. A relationship between these packets may exist due to control messages resulting from the current data transmission.

Another major challenge in traffic fingerprinting is the changes of behavioral patterns in network traffic over time, due to changes in the end-node content. While traffic fingerprinting can be seen as a continuous process with a man-in-the-middle observing network traffic perpetually, a classification model trained initially captures patterns in network traffic available at that particular time. However, traffic patterns may evolve over time, changing their distinguishing characteristics. Since these changes are not reflected in the classifier, its performance degrades while classifying newer data. A recent study in WFIN observed this temporal behavior [23]. Yet, this remains an open challenge.

In this paper, we introduce BIND (fingerprinting with BI-directioNal Dependence), a new set of features from encrypted network traffic, that incorporates feature relationships between consecutive sets of packets in opposite directions. These features are used in conjunction with other independent features to enrich discriminating factors of end-nodes during pattern recognition. Furthermore, we propose a technique for adapting the classifier to temporal changes in data patterns while fingerprinting over a long period of time. Our approach continuously monitors the classifier performance on the training data. When the accuracy drops below a predefined threshold, we replace the classifier with another one trained on the latest data. We call this ADABIND (ADAptive fingerprinting with BI-directioNal Dependence). The summary of our contributions is as follows.

- We propose a new feature extraction method, called BIND, for fingerprinting encrypted traffic to identify an end-node. In particular, we consider relationships among sequences of packets in opposite directions.

- We propose a method, called ADABIND, in which the machine learning classifier adapts to the changes in behavioral patterns that occur when fingerprinting over a long period of time. We continuously monitor classifier performance, and re-train it in an online fashion.

- We evaluate the proposed methods over two applications, namely website fingerprinting (WFIN) and app fingerprinting (AFIN). We perform AFIN over encrypted traffic, which has not been explored in existing studies. Moreover, we use a variety of datasets for both WFIN and AFIN while employing defense mechanisms to show the effectiveness of the proposed approaches especially in the open-world settings.

The rest of the paper is organized as follows. In Section 2, we present relevant background information and related studies in WFIN and AFIN. We present BIND and ADABIND in Section 3. The empirical evaluation including datasets, experiments, and results are detailed in Section 4. Finally, we discuss certain limitations and future work in Section 5 and conclude the paper in Section 6.

2. BACKGROUND

In this section, we present relevant existing studies in traffic analysis, particularly in WFIN and AFIN.

2.1 Website Fingerprinting

The online activity of a user accessing websites can be hidden using anonymity networks such as Tor [14]. Tor provides a low latency encrypted connectivity to the Internet, while anonymizing the connections via a process called pipeline randomization. A circuit of three relay nodes is formed within the Tor network, composed of an entry node, an exit node, and a randomly selected relay node. Circuit connections are reestablished approximately after every 10 minutes of usage [2]. Fingerprinting under this setting is hard due to the decoupling of user request with end-node (i.e., web server) response. Nevertheless, this challenging problem of WFIN has gained popularity in the research community with numerous studies [6,7,21,25,27,36] proposing techniques to perform fingerprinting, and also to defend against it. The inductive assumption is that each website has a unique pattern in which data is transmitted from its server to the user's browser. Moreover, each website content is unique. Using this assumption, the website fingerprinting scenario, generally perceived as an attack against user's privacy, employs a statistical model to predict the website name associated with a given trace. Whereas, a defense mechanism explores methodologies to reduce the effectiveness of such models capable of performing an attack.

2.1.1 Attack

The primary form of attack is to train a classifier using traces collected from different websites, where each trace is represented as a set of independent features. Information present in network packets associated with each trace is summarized to form a histogram feature vector, where the features include packet length (size) and direction (as used in [25]). In addition, Panchenko et al. [28] introduced a set of features extracted from a combination of packets known as *Size Markers or Bursts*. A burst is a sequence of consecutive packets transmitted along the same direction (uplink or downlink). Features such as burst sizes are computed by summing the length of each packet within a burst. These, along with other features such as unique packet sizes, HTML markers, and percentage of incoming and outgoing packets, form the feature vector for a trace. Dyer et al. [17] also used bandwidth and website upload time as features.

A recent work by Panchenko et al. [27] proposes a sampling process on aggregated features of packets to generate overall trace features. Importantly, Cai et al. [7] obtained high classification accuracy by selecting features that involve packet ordering, where the cumulative sum of packet sizes at a given time in each direction is considered. This feature set was also confirmed to provide improved classification accuracy in [36]. It indicates that features capturing relationships among packets in a trace are effective in distinguishing different websites (or end-nodes). In our paper, we focus on extracting such capability from traces in a novel fashion by capturing relationships between consecutive bursts in opposite directions.

While these features are used to train a classifier, e.g. Naïve Bayes [17] and Support Vector Machine (SVM) [28], studies have identified two major settings under which website fingerprinting can be performed. First, the user is assumed to access only a small set of known websites. This restriction simplifies the training process since the attacker can train a model in a supervised manner by considering traces only from those websites. This form of classification is known as *closed-world*. However, such a constraint is not valid in general as a user can have unrestricted access to a large number of websites. In this case, training a classifier by collecting trace samples from all websites to perform multi-class classification is unrealistic. Therefore, an adversary is assumed to *monitor* access to a small set of websites called the *monitored set*. The objective is to predict whether a user accesses one of these monitored websites or not. This binary classification setting is called *open-world*. Wang et al. [36] propose a feature weighting algorithm to train a k-Nearest Neighbor (k-NN) classifier in the open-world setting. They utilize a subset of traces from the monitored websites to learn feature weights which are used to improve classification. In this paper, we evaluate our proposed feature extraction approach on both these settings. Particularly for the open-world case, we utilize the feature weighting method proposed in [36] to perform a comparative study of feature extraction techniques.

A study by Juarez et at. [23] observes and evaluates various assumptions made in previous studies regarding WFIN. These include page load parsing by an adversary, background noise, sequential browsing behavior of a user, and replicability due to staleness in training data with time, among others. While recent studies [18,38] have addressed each of these issues by relaxing appropriate assumptions, the issue of replicability still remains an open challenge. Wang et al. [38] attempt to address the issue of staleness in training data over time within their k-NN model [36] specific to open-world. They score the training data consisting of traces based on model performance of 20 nearest neighbors. However, this methodology cannot be generalized, i.e., it is not applicable if one uses a classifier other than k-NN. Moreover, it is also not applicable to the closed-world setting. In this paper, we introduce a generic method to update the classifier model for replicability of WFIN and AFIN over long periods of time.

2.1.2 Defense

Since a successful attack depends on the characteristic network packet features used to train a model, defenses against WFIN involve disguising these features to reduce distinguishing patterns in network traces. Such defense mechanisms vary from padding packets with extra bytes, to morphing the website packet length distribution such that it appears to come from another target distribution (i.e., a different website) [17]. In packet padding, each packet size in the trace is increased to a certain value depending on the padding method used. These methods include *Pad-to-MTU* [17], *Direct Target Sampling* (DTS), and *Traffic Morphing* (TM) [40].

Pad-to-MTU pads each packet to the maximum size limit in TCP protocol (Maximum Transmission Unit or MTU). With all packet sizes equal, use of the packet length feature for obtaining deterministic patterns might be less effective. However, this method is not widely used in practice as it increases network latency and incurs high overhead when most of the packets in a trace are of length less than MTU. Nevertheless, early studies [25] showed that attacks with considerable success are possible even when defenses like packet padding are used.

This led to a study in [40] that introduced more sophisticated distribution-based padding methods such as DTS and TM. In DTS, using random sampling, the distribution of the packet length in a trace belonging to a website is made to appear similar to the packet length distribution of another website. This requires less overhead than Pad-to-MTU. TM further improves DTS by using a cost minimization function between two websites to minimize packet padding, while maximizing similarity between them. In our study, we evaluate BIND by applying these padding techniques to packets while performing the closed-world settings in website fingerprinting.

In the case of open-world setting, Dyer et al. [17] introduced a defense mechanism, called Buffered Fixed Length Obfuscator (or *BuFLO*), that not only uses packet padding, but also modifies packet timing information by sending packets in fixed intervals. Cai et al. [6] improved BuFLO and introduced a lighter defense mechanism, called *Tamaraw*, which considers different time intervals for uplink and downlink packets in the open-world setting. We utilize these mechanisms in the open-world setting to evaluate BIND.

2.2 App Fingerprinting

An increase in popularity of smartphone applications has attracted researchers to study the issues of user privacy and data security in apps developed by third-party developers [35]. In particular, many studies have proposed methods to perform traffic analysis while a user uses an app. Dai et al. [12] first proposed a method to identify an app by using the request-response mechanisms of API calls found in HTTP packets. They perform UI fuzzing on apps whose network packets are captured using an emulator. Similarly, [26] proposes a method to fingerprint apps using comprehensive traffic observations. These studies perform app identification (or fingerprinting) using only HTTP traffic. Such methods cannot be applied on HTTPS traffic since the packet content is encrypted and not readily available.

Studies on performing traffic analysis over HTTPS app network traffic explore varied applications including smartphone fingerprinting [33], user action identification [9,10], user location tracking [3], and app identification [26]. They use packet features such as packet length, timing information, and other statistics to build classifiers for identification (or prediction). Note that this is similar to the WFIN setting mentioned in §2.1. Recently, a study [34] performed AFIN using both HTTP and HTTPS data. They use features such as burst statistics and network flows. Here, a flow is a set of network packets belonging to the same TCP session. They train a random forest classifier (ensemble of weak learners) and a support vector machine (SVM) using features extracted from network traffic of about 110 apps from the Google play store. Evaluation of their method is similar to the closed-world setting of WFIN, where network traffic from apps considered for training and testing the model belong to a closed set, i.e., the user has access to only a finite known set of apps. The method resulted in an overall accuracy of 86.9% using random forest, and 42.4% using SVM. These results are based on a small dataset of apps which may have both HTTP and HTTPS traffic. Furthermore, they only show a closed-world setting. However, with a large number of apps present on various app stores, these results may not reflect a realistic scenario of the open-world setting in AFIN.

Similar to that of WFIN, the open-world setting in AFIN assumes that the man-in-the-middle monitors the use of a small set of apps called the *monitored set*. The goal is to determine whether a user is running an app that belongs to this set. In our evaluation, we use our proposed technique for traffic analysis on a larger dataset of apps that only use HTTPS for connecting to remote services. Contrary to WFIN where the network is anonymized, apps do not use an anonymity network. However, the effect of anonymization is similar to that of WFIN. In WFIN, anonymization results in removal of destination website identifiers (i.e., IP address). In AFIN, apps connect to multiple remote hosts deriving remote services from them. However, multiple apps may connect to the same host. A mere list of hosts or IP addresses is not sufficient to deterministically identify an app. This property effectively anonymizes such apps with respect to the network. We therefore rely on traffic analysis to perform AFIN. In this paper, we

Category	Features
Packet (Up/Dn)	Packet length
Uni-Burst (Up/Dn)	Uni-Burst size
	Uni-Burst time *
	Uni-Burst count
Bi-Burst (Up-Dn/Dn-Up)	Bi-Burst size *
	Bi-Burst time *

*new features introduced in this paper

Table 1: Features from Packets, Uni-Bursts, and Bi-Bursts.

show the applicability of both closed-world and open-world settings while utilizing the BIND feature extraction method.

3. PROPOSED APPROACH

In this section, we present the methodology to extract the BIND features, and detail the ADABIND approach.

3.1 Features

With encrypted payload of each packet in a trace, we extract features from packet headers only. The main idea is to extract features from consecutive bursts to capture any *dependencies* that may exist between them. As illustrated in Figure 2, we call the burst directed from a user/client (or app) to server (e.g., burst a), an uplink uni-burst (or *Up uni-burst*), and the burst directed from server to the user, a downlink uni-burst (or *Dn uni-burst*) (e.g., burst b). Similar to packets, a burst or uni-burst has features such as size (or length), time, and direction. Uni-burst size is the summation of lengths of all its packets. Packet time is the departure/arrival timestamp in the uplink/downlink direction, measured near the user-end of the network by a man-in-the-middle. Uni-burst time is the difference between the last packet's timestamp and the first packet's timestamp within a burst, i.e., the time taken to transmit all packets of a burst in a specific direction. Here, the term burst and uni-burst are equivalent. The name uni-burst emphasizes on the fact that features are extracted from a single burst, as opposed to *Bi-Burst* which is a tuple formed by a sequence of two adjacent uni-bursts in opposite direction (e.g., burst b and c in Figure 2).

Bi-Burst features. Features extracted from Bi-Bursts are as follows.

1. **Dn-Up-Burst size**: Dn-Up-Burst is a set of tuples formed by downlink (Dn) - uplink (Up) consecutive bursts. Here, unique tuples are formed according to the corresponding uni-burst lengths where each tuple forms a new feature.

2. **Dn-Up-Burst time**: This set of features considers unique consecutive uni-burst time tuples between adjacent Dn uni-burst and Up uni-burst sequences.

3. **Up-Dn-Burst size**: Similar to Dn-Up-Burst size features, these features consider burst length tuples of adjacent Up uni-burst and Dn uni-burst sequences.

4. **Up-Dn-Burst time**: Similar to Dn-Up-Burst time features, this set of features considers burst time tuples formed by adjacent Up uni-burst and Dn uni-burst sequences.

In each trace, we count such unique tuples to generate a set of features. To overcome dimensionality issues associated with burst sizes, *quantization* [15] is applied to group bursts into correlation sets (e.g., based on frequency of occurrence).

Packet and Uni-Burst features. In addition to the Bi-Bursts features, we also use burst size and burst time features. Previous studies [17] only consider total trace time as a feature, contrary to the burst time feature we use in this paper. Furthermore, we also consider the count of packets within a burst as a feature. In order to capture variations of the packet features, we use an array of unique packet lengths as well. The set of features, termed as BIND, are listed in Table 1. All these features are concatenated to form a large array of features (*histograms*) to be extracted from each trace. A set of multiple traces represented in this manner forms the training and testing set.

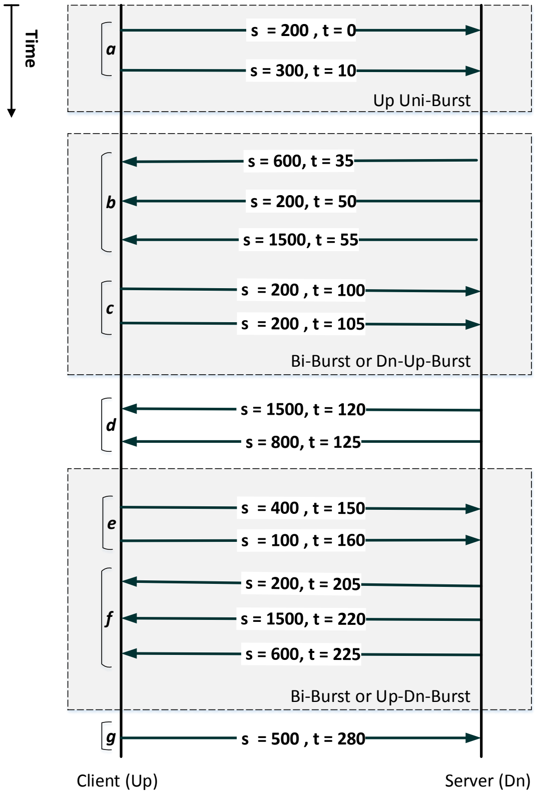

Figure 2: An example illustrating BIND Features.

Example. Figure 2 depicts a simple trace where packet sequences between uplink and downlink are shown. Each packet in the figure has size s in bytes and time t in milliseconds. We set time for the first packet in the trace to zero, as a reference. An example of a uni-burst is shown as burst a, whose size is 500, computed by adding packet sizes $s = 200$ and $s = 300$ that form the burst. Its time is computed as 10, which is the absolute time difference between the last packet ($t = 10$) and the first packet ($t = 0$) in the burst. Similarly, a Bi-Burst example is shown as well, formed with a combination of bursts b and c. This is denoted as Dn-Up-Burst. In this case, the Bi-Burst tuple using the burst size (i.e., Dn-Up-Burst size) is represented as {DnUp_2300_400}, where 2300 is the burst size of b, and 400 is the burst size of c. We count the number of such unique tuples in the trace. In this case, the count for {DnUp_2300_400} is 1.

3.2 Learning

In the closed-world setting, we use the BIND features to train a support vector machine (SVM) [11] classifier. SVM applies convex optimization and maps non-linearly separated data to a higher dimensional linearly separated feature space. Whereas in the open-world setting, using the BIND features, we apply the weighted k-Nearest Neighbor (k-NN) approach proposed in [36]. Feature weights are computed using traces from the monitored set. During testing of traces with

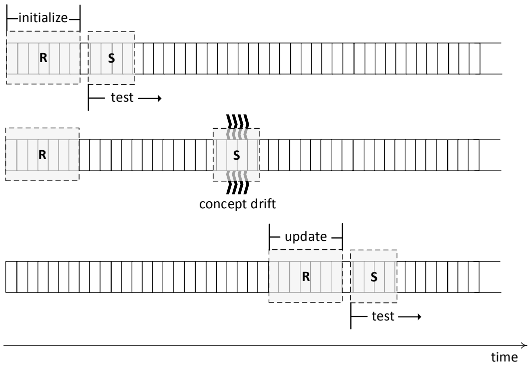

Figure 3: Illustration of AdaBind.

unknown class labels, these feature weights are applied. Majority class voting among k-Nearest Neighbors is performed to predict class label of a test trace. Additionally, we also use a Random Forest classifier in the open-world setting. Instead of performing feature weighing, which is computationally expensive, we use a set of weak learners to form an ensemble of decision trees (random forest).

3.2.1 *Static Learning*

Typically, previous studies (mentioned in §2.1) have focused on performing fingerprinting by collecting traces for a short period of time. Classifiers are trained on traces collected within this time period, and used to predict class labels thereafter. We refer to this type of classifier training as static. On the contrary, WFin and AFin can be viewed as a continuous process involving trace collection over a long period of time. Moreover, data collection is time consuming. Changes in data content transmitted between end-nodes affect patterns captured in the model. Using a static model to predict class labels of test traces in this situation drastically affects classification performance.

3.2.2 *Adaptive Learning*

We now present the details of AdaBind. In this section, we show how we model encrypted data fingerprinting in an adaptive manner. As discussed in §3.2.1, over time, the data patterns of the current traces may be different from the patterns in previously seen training traces. This is known as *concept drift* [19,20]. To address this challenge, the model has to be updated (re-trained) regularly. We study the effect of re-training as follows.

Fixed update. One simple approach is to apply fixed updates to re-train the model periodically. We refer to this approach as BindFup (BIND Fixed UPdate). BindFup updates the model periodically, regardless of any concept drift that may happen. The model will be re-trained regularly (e.g., at the end of every week) with freshly obtained training data. There are two possible scenarios, *early update* and *late update*. In early update, BindFup updates the model in a way that ensures no concept drift in data. Although this update is more accurate and stable, it may suffer from unnecessary re-training which will add significant overhead to the classification process. On the other hand, late update may miss possible concept drift in data over time which affects the overall performance of the model.

Dynamic update. In this approach, as depicted in Figure 3, we update the model whenever there is a drift between the current data and previously seen training data. R is a training window that builds the model, while S is a sliding window that probes this model for any possible concept drift (i.e., model needs update). Algorithm 1 describes this dynamic update mechanism. We refer to this algorithm as BindDup (BIND Dynamic UPdate). BindDup starts by considering a portion of data as a training window to initialize the AdaBind model (lines 2 and 3). Then, the subsequent instances are considered within a sliding window to validate the performance of this model over time (lines 5 and 6). If the accuracy drops below a predefined threshold (line 7), the initial AdaBind model becomes obsolete (i.e., concept drift) and the training window moves (line 8) to get new instances to re-train and update the model (lines 9 and 10). BindDup utilizes the AdaBind updated model to test incoming new data in a continuous fashion.

Algorithm 1: BindDup

```
   Data: Training Data: $TrainX$, Testing Data: $TestX$
   Input: Training Window: $\mathcal{R}$, Sliding Window: $\mathcal{S}$, Threshold: $\mathcal{T}$
1  begin
2      $\mathcal{F}^{train} \leftarrow extractFeatures(TrainX^{\mathcal{R}})$;
3      $initializeModel(\textsc{AdaBind}, \mathcal{F}^{train})$;
4      for each $\mathcal{S}$ do
5          $\mathcal{F}^{test} \leftarrow extractFeatures(TestX^{\mathcal{S}})$;
6          $accuracy \leftarrow validateModel(\textsc{AdaBind}, \mathcal{F}^{test})$;
7          if $accuracy < \mathcal{T}$ then
8              $move\ \mathcal{R}$;
9              $\mathcal{F}^{train} \leftarrow extractFeatures(TrainX^{\mathcal{R}})$;
10             $updateModel(\textsc{AdaBind}, \mathcal{F}^{train})$;
11             $move\ \mathcal{S}$;
12         end
13     end
14 end
```

Dataset	# of websites		# of traces per website
Https [25]	Monitored	30	70
	Non-Monitored	970	1
Tor [36]	Monitored	100	90
	Non-Monitored	5000	1

Table 2: Statistics for Website Fingerprinting datasets in the open-world setting.

4. EVALUATION

In this section, we present the empirical results of using BIND for WFin and AFin, comparing it with other existing methods.

4.1 Datasets

We use two existing datasets for evaluating WFin, one using HTTPS and the other using the Tor anonymity network, referred to as Https and Tor respectively. These datasets have been widely used in previous research on traffic fingerprinting. For AFin, we collect our own dataset from apps that use the HTTPS protocol.

Website Datasets. The first dataset presented in [25], which we denote as Https, was collected while browsing websites using the HTTPS protocol along with a proxy server to imitate an anonymity network. The authors followed a ranking procedure to select the most accessed websites in their school department. The second dataset is described in [36]. This dataset is collected by capturing packets generated from a browser connected to the Tor anonymity network. We denote this dataset as Tor.

Https consists of 1000 websites with 200 traces each. For WFin, we evaluate the closed-world setting by randomly picking a subset of these 1000 websites. For the open-world setting, we randomly select 30 websites as the monitored set, and the rest as the non-monitored one.

The other dataset (Tor) consists of two sets of traces. The first is a set of 100 websites that have 90 traces each. These websites were selected from a list of blocked websites by some countries. We use this for the closed-world experiments. The second set consists of 5000 websites that have one trace each. These websites were selected

Category	# of apps		# of traces per app
APP-FIN	Monitored	30	20
	Non-Monitored	2238	1
APP-COMM	Non-Monitored	1061	1
APP-SOCIAL	Non-Monitored	1290	1

Table 3: Dataset statistics for App Fingerprinting in the open-world setting.

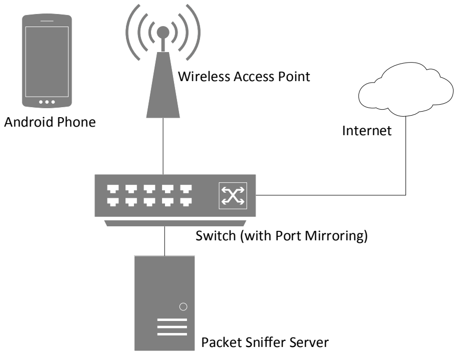

Figure 4: Illustration of the app trace data collection process

from Alexa's top websites [1]. In the open-world setting, we use the set of 100 websites as monitored, and the set of 5000 websites as non-monitored. The summarized statistics of these datasets are provided in Table 2. These two datasets enable us to perform an unbiased comparison of BIND with other competing methods.

App Dataset. For AFIN, we evaluate BIND using a dataset that we collected by executing multiple Android apps on a Samsung Galaxy S device, running Android version 4.3.1. We randomly select about 30,000 apps from three different categories in *Google Play Store*. The categories include Finance, Communication, and Social. We refer to them as APP-FIN, APP-COMM, and APP-SOCIAL respectively. We then install and launch these apps on the phone which is connected to the Internet via a wireless router. Each trace per app is collected over a 30-sec period passively using a mirroring switch at the wireless router. Figure 4 illustrates this data collection setup. We filtered the captured traffic to contain packets from ports 80, 8080, and 443. We then identify apps that use only HTTPS data from the captured traces. These traces from such apps are then used to perform the closed-world and open-world AFIN. It is important to note that we uninstall each app as soon as we complete capturing a trace to avoid any background noise during further trace generation.

Similar to WFIN, multiple traces of apps are required to train a classifier in the closed-world and open-world settings. We use the APP-FIN dataset for performing the closed-world experiments as we capture multiple traces for each app. We only capture a single trace per app for APP-COMM and APP-SOCIAL to be used for the open-world experiments as the non-monitored set. The dataset statistics for the open-world setting are shown in Table 3. Note that in the closed-world setting, we only evaluate using apps from APP-FIN. In the case of open-world, the monitored apps are considered only from APP-FIN and the non-monitored apps are considered from all categories shown in Table 3.

While performing app selection for creating our dataset, we observed a few interesting statistics that would further motivate the problem of AFIN. Figure 5 shows the percentage of apps that use HTTP and HTTPS data at launch in our initial set of $30,000$ apps. Observe that most apps use HTTP along with HTTPS while a sizable portion of apps use only HTTPS, for communication over the Internet. Furthermore, we obtained a list of IP addresses from HTTPS apps in each category We found a total of 1115 unique IP addresses for APP-FIN, 820 for APP-COMM, and 900 for APP-SOCIAL. Additionally, each app connects to 3 different IP addresses on average over the whole dataset. This clearly indicates that the IP addresses found on HTTPS traffic overlap across apps, and do not provide sufficient information to identify the app generating a trace by itself.

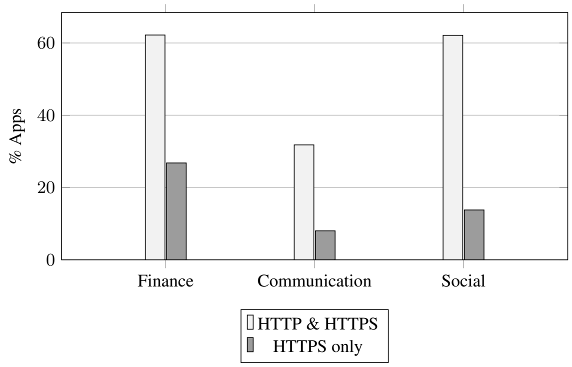

Figure 5: Empirical Statistics of Android Apps

Data Analysis Method	Setting Type	Features	Classifier
VNG++ [17]	Closed	Uni-Burst Size & Count Total Trace Time Uplink/Downlink Bytes	Naïve Bayes
P [28]	Closed	Uni-Burst Size & Count Packet Size Packet Ordering	SVM
OSAD [37]	Closed	Cell Traces	Optimized SVM
BINDSVM *	Closed	BIND features: Bi-Burst Size & Time Uni-Burst Size, Time, & Count Packet Size	SVM
WKNN [36]	Open	Same features as P	Weighted k-NN
BINDWKNN *	Open	BIND features: Same features as BINDSVM	Weighted k-NN
BINDRF *	Open	BIND features: Same features as BINDSVM	Random Forest

*new approaches introduced in this paper

Table 4: Traffic Analysis Techniques used for the evaluation

4.2 Experimental Settings

Using these datasets, we perform our analysis on both closed-world and open-world settings. For a comparative evaluation, we consider existing traffic analysis techniques developed for WFIN. These techniques are listed in Table 4. The table details the features and classifiers used for our evaluation in both Closed-world (Closed) and Open-world (Open) settings. For brevity of representation, we term websites (in the case of WFIN) or apps (in the case of AFIN) as *entities*.

Closed-world. Using BIND features, we use a support vector machine classifier (SVM) in the closed-world setting. We refer to this approach as BINDSVM as shown in Table 4. In our experiments, we use a publicly available library called LibSVM [8] with a Radial Basis Function (RBF) kernel having the parameters $Cost = 1.3 \times 10^5$ and $\gamma = 1.9 \times 10^{-6}$ (following recommendations in [28]). We consider varied subsets of entities to evaluate the feature set. Particularly, we use 16 randomly selected traces per entity (class) for training a classifier, and 4 randomly selected traces per entity for testing. For each experiment, we chose the number of selected (monitored) entities in $\{20, 40, 60, 80, 100\}$.

Open-world. For the open-world scenario, as discussed in §3.2, we use two classification methods with the BIND features. First, we use the weighted k-NN mechanism proposed in [36]. Specifically, we use $k = 1$ since it is shown to produce the best results on the TOR dataset in [36]. We denote this method as BINDWKNN as shown in Table 4. Furthermore, we also use the Random Forest classifier with BIND features, denoted as BINDRF in Table 4. We use a set of 100 weak learners to form an ensemble of decision trees. We use the scikit-learn [29] implementation for our evaluation. The complete set of monitored and non-monitored traces mentioned in Tables 2 and 3 are considered for evaluation.

Evaluation Measure. The results of the closed-world evaluation are measured by computing the average accuracy of classifying the correct class for all test traces. We randomly select traces from the corresponding dataset and repeat each experiment 10 times with different entities and traces. Average accuracy is computed across these experiments. In the open-world evaluation, we measure the true positive rate (TPR) and false positve rate (FPR) of the binary classification. These are defined as follows: $TPR = \frac{TP}{TP+FN}$ and $FPR = \frac{FP}{FP+TN}$. Here, TP (True Positive) is the number of traces which are monitored, and predicted as monitored by the classifier. FP (False Positive) is the number of traces which are non-monitored, but predicted as monitored. TN (True Negative) is the number of traces which are non-monitored and predicted as non-monitored. FN (False Negative) is the number of traces which are monitored, but predicted as non-monitored. We perform a 10-fold cross validation on each dataset, which gives randomized instance ordering.

In order to evaluate the performance of BIND against defenses discussed in §2.1, we consider one of the most sophisticated and complex defenses, Traffic Morphing (TM). Furthermore, to evaluate BIND against existing approaches, for the open-world setting on the TOR dataset, we apply the *Tamaraw* defense mechanism, designed specifically for Tor, as evaluations in [6,36] show that this defense performs exceptionally well against TOR.

4.3 Experimental Results

We use the notations given in Table 2 and Table 3 to denote the WFIN and AFIN datasets respectively.

4.3.1 Traffic Analysis

We first perform WFIN and AFIN experiments in the closed-world setting. Here, a set of randomly chosen entities are classified using competing methods. We vary the set size from 20 to 100. The results are presented in Table 5 using the HTTPS and TOR datasets for WFIN, and the APP-FIN dataset for AFIN. In some cases, we can see BINDSVM performs comparatively closer to or lower than the other competing methods, while outperforming them in other cases. For example, with 80 websites considered, the average accuracy of BINDSVM (BIND using SVM) on the HTTPS dataset is 88.4%. This is marginally greater than 88.3% obtained from the P method. Similarly in AFIN, BIND resulted in an average accuracy of 87.8%, compared to a marginally better accuracy of 88% resulting from the P method. Moreover for the TOR dataset, it is not surprising that the OSAD method performs the best in all experimental settings since it uses a distance measure that is specifically applicable to Tor data. In the closed-world setting, most methods listed in Table 4 use features that overlap or hold similar information about the class label. Some features provide better characteristic information about the class than others. When selecting the websites at random during evaluation, each classification method outperforms the other in a few cases depending on the data selected for training and testing. Therefore, the average accuracy across these are marginally superior than others in most of the cases.

However, the greatest impact of using BIND features can be observed in the more realistic open-world setting. Table 6 presents the results of the open-world setting for all competing methods. Here, a high value of TPR and a low value of FPR are desired. As mentioned earlier in this section, we use two types of classifiers while using the BIND features, i.e., BINDWKNN and BINDRF. In the case of WFIN, it is clear that the TPR for both BINDWKNN and BINDRF is significantly better compared to that of WKNN. For instance, consider the result of the TOR dataset. The TPR obtained from BINDWKNN method is 90.4% and that obtained from BINDRF is 99.8%, as compared to 89.6% of WKNN. The BINDRF method outperforms WKNN even though the WKNN method was specifically designed for high quality results on this dataset. In terms of FPR, BINDWKNN method performs better than WKNN.

A more significant result can be observed in the open-world setting of AFIN. Both TPR and FPR are greatly improved with the BINDWKNN and BINDRF methods on all app fingerprinting datasets, as indicated in Table 6. For example, the average TPR resulting from BINDWKNN method on the APP-FIN dataset is 78%, compared to the average TPR of 53% reported by the WKNN method. Similarly, the average FPR of 7% reported by the BINDWKNN method is better than the average FPR of 10% resulting from the WKNN method. This clearly demonstrates the effectiveness of using BIND features for traffic analysis in AFIN as well.

Moreover, the average TPR and FPR are largely improved when using the BINDRF method. It is important to note that while using monitored and non-monitored traces from different categories, i.e., in the case of the APP-COMM and APP-SOCIAL datasets, the average TPR and FPR are better when compared with the results from the APP-FIN dataset where the monitored and non-monitored sets are from the same category. Especially, a low FPR of less than 1% is obtained on these datasets. This indicates that there exist differentiating characteristics between apps from different categories as expected.

The open-world setting is a binary classification problem. Features extracted and the classifier used for determining class boundary significantly impact the TPR and FPR results. In the case of WKNN, the monitored entities are made as close as possible via an iterative weighing mechanism. When using BIND features, we count unique bi-burst tuples. These provide additional features to the existing feature set of uni-burst used in [36]. These features aid the weighing mechanism by bringing out more relevant dimensions, suppressing less relevant ones in BINDWKNN. Random forest uses decision trees that divide the feature space effectively using the information gain measure rather than the Euclidean distance measure used by the k-NN method. An ensemble of such classifiers typically reduces bias and variance during training, compared to a single classifier [5]. Consequently, this classifier, along with BIND features, shows superior performance in TPR results.

4.3.2 Traffic Analysis with Defenses for Website Fingerprinting

We now consider the evaluation of BIND in an adversarial environment, specifically for WFIN, similar to relevant studies in this area. Here, we apply a defense mechanism to trace packets for with the aim of reducing effectiveness of a fingerprinting attack (classifier), and study the robustness of BIND when used by an attacker against such defenses.

With defense mechanisms such as Traffic Morphing (TM) used by defenders to thwart classifiers, the features extracted from the data play an important role while performing an adversarial attack. Table 7 shows

Dataset		HTTPS				TOR				APP-FIN			
Method		VNG++	P	OSAD	BINDSVM	VNG++	P	OSAD	BINDSVM	VNG++	P	OSAD	BINDSVM
# entities	20	87.5	93.5	**94.1**	94.0	78.0	85.3	**90.0**	86.5	81.3	92.0	88.7	**93.3**
	40	83.8	**91.4**	89.0	91.3	67.8	77.6	**92.1**	80.9	73.6	**88.3**	85.1	87.3
	60	85.2	**92.3**	91.0	91.6	63.7	77.0	**86.7**	79.5	72.3	86.5	83.6	**86.7**
	80	81.6	88.3	87.7	**88.4**	62.9	75.8	**89.5**	77.6	72.8	**88.0**	79.6	87.8
	100	82.4	**90.3**	89.2	90.0	56.9	71.4	**85.7**	73.9	66.0	83.1	77.2	**84.2**

Table 5: Accuracy (in %) of the closed-world traffic analysis for website fingerprinting (HTTPS and Tor) and app fingerprinting (App-Finance) without defenses.

Dataset	HTTPS			TOR			APP-FIN			APP-COMM			APP-SOCIAL		
Method	WKNN	BINDWKNN	BINDRF	WKNN	BINDWKNN	BINDRF	WKNN	BINDWKNN	BINDRF	WKNN	BINDWKNN	BINDRF	WKNN	BINDWKNN	BINDRF
TPR	73.0	91.0	**98.2**	89.6	90.4	**99.8**	53.0	78.0	**88.5**	64.0	82.0	**93.1**	61.0	75.0	**92.1**
FPR	29.0	**16.0**	18.3	2.1	**1.9**	3.4	10.0	7.0	**1.9**	5.0	2.0	**0.8**	5.0	2.0	**0.1**

Table 6: TPR and FPR (in %) of open-world setting for website fingerprinting (HTTPS and Tor) and app fingerprinting (App-Finance, App-Communication and App-Social) without defenses.

Dataset		HTTPS			
Method		VNG++	P	OSAD	BINDSVM
# websites	20	79.1	76.0	86.6	**87.5**
	40	74.4	73.6	79.1	**82.6**
	60	68.4	68.0	74.6	**79.7**
	80	61.2	65.1	69.8	**75.2**
	100	64.1	60.6	67.4	**73.2**

Table 7: Accuracy (in %) of closed-world website fingerprinting on HTTPS dataset with Traffic Morphing.

Dataset		TOR			
Method		VNG++	P	OSAD	BINDSVM
# websites	20	77.8	81.3	68.5	**82.3**
	40	66.6	74.9	58.5	**77.6**
	60	61.0	70.3	51.2	**72.3**
	80	58.7	67.6	42.6	**69.9**
	100	65.8	65.8	39.3	**68.7**

Table 8: Accuracy (in %) of closed-world website fingerprinting on Tor dataset with Traffic Morphing.

Dataset	Score	Method		
		WKNN	BINDWKNN	BINDRF
HTTPS	TPR	74.0	82.0	**98.5**
	FPR	29.0	**24.0**	72.4
Tor	TPR	2.7	2.7	**100.0**
	FPR	0.0	0.0	0.0

Table 9: TPR and FPR (in %) in open-world setting for website fingerprinting on HTTPS dataset with Traffic Morphing, and Tor dataset with Tamaraw.

the average accuracy obtained on the HTTPS dataset when TM is applied on all websites in the closed-world setting. It is important to note that for every experiment, we apply TM by selecting a random target website. BINDSVM performs with significant improvement in average accuracy on all experiment settings compared to other competing methods. For instance, BINDSVM reports an average accuracy of 73.2% with 100 closed-world websites. This is better than the average accuracy of 67.4% reported by OSAD, which is the second highest accuracy in this setting.

Similarly, Table 8 shows the average accuracy obtained on the TOR dataset when TM is applied on all websites. From the table, we can observe that the BINDSVM method outperforms other methods.

In the open-world setting, we apply TM on the HTTPS dataset. The TPR and FPR results are shown in Table 9. The BINDRF method reports an average TPR of 98.5%. However, it also reports an undesirable high FPR of 72.4%. This high FPR indicates that more false alarms are reported by this classifer. In contrast, the BINDWKNN method reports 82% average TPR, which is greater than 74% reported by the WKNN method. Moreover, it also reports the lowest average FPR of 24% on the dataset. This shows the effectiveness of this defense on HTTPS dataset. It also indicates that BIND features aid the weighted k-Nearest Neighbors algorithm to classify more accurately than merely using Uni-Burst features.

Table 9 also shows the average TPR and FPR obtained on the TOR dataset when using competing methods while applying the Tamaraw defense mechanism. In the case of methods that use the weighted k-NN algorithm, i.e., WKNN and BINDWKNN, we obtain a low TPR of 2.7%. This result agrees with that reported by Wang et al. [36] who use the WKNN method on the same dataset. Yet rather remarkably, we obtain an average TPR of 100% and an average FPR of 0% from the BINDRF method. This highly accurate classification is a result of a combination of BIND features and random forest classifiers, where features of monitored websites are morphed by Tamaraw. Moreover, the morphing scheme involves changing packet time and size values. In the BIND feature set, we consider quantized tuple counts as features (Bi-Burst), along with other Uni-Burst features. Changing the packet time information by a constant may not successfully destroy characteristic information in a trace. Furthermore, the tree structure of weak learners (decision trees) in the random forest classifier aids in a better classification as illustrated in Table 6. This combination provides a perfect classification of the morphed dataset in this case.

4.3.3 *Traffic Analysis with Defenses for App Fingerprinting*

We evaluated our proposed data analysis technique in an adversarial environment for WFIN. A user may visit any website s/he desires using an anonymity network to protect against surveillance from external adversaries on the network. However, this case may not be directly applicable to AFIN. An app is typically deployed on a well-recognized app store such as Google play. These apps typically may not provide users an ability to configure network traffic to use a user-desired anonymity network such as Tor. They use the default network configuration set on the host device. However, the goal of an adversary in AFIN might be to identify vulnerable apps or malware installed on a device in order to perform attacks such as privilege escalation [13] targeted on the user. Therefore, we perform experiments on app traffic when defenses such as TM are applied to reduce chances of app identification.

We assume that defenses like packet padding could be applied to app traffic and evaluate the data analysis techniques when the padding technique of TM is used. Instead of morphing the packet distribution of a website with another one in the case of WFIN, packet distribution of

Dataset		App-Fin			
Method		VNG++	P	OSAD	BindSvm
# Apps	20	71.5	68.3	**77.6**	77.0
	40	58.3	59.1	61.0	**67.0**
	60	50.2	51.7	56.0	**59.2**
	80	44.6	44.8	49.3	**53.8**
	100	42.9	42.1	49.2	**50.4**

Table 10: Accuracy (in %) of closed-world app fingerprinting while using Traffic Morphing.

Dataset	Score	Method		
		Wknn	BindWknn	BindRF
App-Fin	TPR	16.0	**22.0**	20.5
	FPR	14.0	13.0	**5.1**
App-Comm	TPR	41.0	46.0	**66.8**
	FPR	7.0	5.0	**4.1**
App-Social	TPR	67.0	68.0	**68.6**
	FPR	5.0	4.0	**1.2**

Table 11: TPR and FPR (in %) of open-world app fingerprinting while using Traffic Morphing.

an app is morphed to appear similar to another app. Table 10 shows the accuracy of this scenario in the closed-world setting on the App-Fin dataset with the morphed traffic. Similar to the results in Table 7, the average accuracy reported by BindSvm method is higher than other competing methods in most cases. Results of the open-world setting are given in Table 11. Clearly, BIND performs better than other competing methods. A low FPR with a high TPR are reported by the BindRF method compared to Wknn. Another important observation is that the TPR resulting from the App-Fin dataset is lower than other categories. This shows that intra-category differentiating characteristic features may be affected more than inter-category features while using morphing techniques. Overall, these results reinforce our hypothesis that BIND methods provide good characteristic properties from traces which can be used for better entity identification.

However, we realize that TPR is low when compared to that of the Wfin datasets in Table 9. The network signature of an app is different from that of a website. Apps use the Internet to connect to services and communicate minimal amount of data as necessary. In contrast, browsing a website could potentially generate a larger network trace since all the components of a website have to be downloaded to the browser. A smaller network footprint may affect the fingerprinting process.

4.3.4 Execution Time

Figure 6 shows the execution time for experiments in Table 5 on the Tor dataset, where OSAD outperforms the other methods. The x-axis in the figure represents the number of websites, while the y-axis represents the execution time (in seconds) in logarithmic scale (base 10). The execution times of VNG++, P, and BindSvm classifiers are low compared to that of OSAD. For instance, with 60 websites, OSAD takes 2340 sec while VNG++, P, and BindSvm take 25, 31, and 39 sec, respectively. This shows how OSAD incurs extra overhead which may render it impractical in some scenarios. In the case of open-world setting, we observed that Wknn and BindWknn ($>$ 30 mins) took significantly longer time than BindRF ($<$ 60 secs), due to weight computations. Yet, BindRF outperformed BindWknn (or Wknn) in Table 6 and Table 11 on most cases.

4.3.5 Base Detection Rate Analysis

In this section, for the open-world scenario, we study the effect of BIND in a more realistic scenario which considers the probability of a client visiting a website or using an app in the monitored set, referred to as *prior* or *base rate*. This has been recently raised as a concern in the research community in Wfin [23].

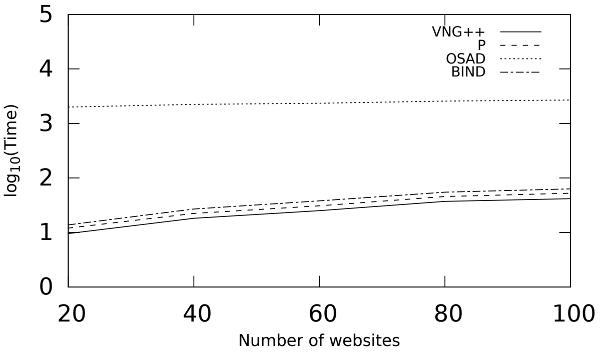

Figure 6: Running time (in seconds) for the experiments in Table 5, on TOR dataset. Note that time axis is in logarithmic scale to the base 10.

The *base detection rate* (*BDR*) is the probability of a trace being actually monitored, given that the classifier predicted (detected) it as monitored. Using the Bayes Theorem, *BDR* is formulated as:

$$P(M|D) = \frac{P(M)\ P(D|M)}{P(M)\ P(D|M) + P(\neg M)\ P(D|\neg M)]}, \quad (1)$$

where M and D are random variables denoting the actual monitored and the detection as monitored by the classifier, respectively. We use TPR and FPR, from Table 6, as approximations of $P(D|M)$ and $P(D|\neg M)$, respectively.

Table 12 presents the *BDR* computed for the open-world classifiers. We assume $P(M)$ or *prior* is calculated as the size of the monitored set divided by the world size (the size of the monitored and non-monitored set), i.e., $P(M) = \frac{|monitored|}{|monitored|+|non-monitored|}$. The table shows the *BDR* for the different datasets.

Although BIND methods ourperform other methods, as the results in Table 12 indicate, the numbers expose a practical concern in fingerprinting research: despite having high accuracy values, typical fingerprinting detection methods are rendered ineffective when confronted with their staggeringly low base detection rates. This is in part due to their intrinsic inability to eliminate false positives in operational contexts.

However, we follow a similar approach to the results of a recent study [16] in Anomaly Detection to approximate the prior for the specific scenario of a *targeted user*. The study assumes a model with a determined attacker leveraging one or more exploits of known vulnerabilities to penetrate a typical organization's internal network, and approximates the *prior* of a directed attack to 6% (using threat statistics from 2011). Similarly, we model a targeted user where the *prior* increases given other estimates. For example, consider a government tracking a suspicious user (targeted) with a prior knowledge or estimate that increases the probability of such user visiting certain websites or using certain apps (monitored) or carrying out specific online activities (e.g. suspicious activities).

Figure 7 depicts this process using TPR and FPR obtained from Table 6 with the Tor dataset. In this figure, we show the effect of increasing the *prior*, starting from 2% which is the actual $P(M)$. Similarly, Figure 8 shows the effect of increasing this *prior* on the same dataset while applying the Tamaraw defense, using TPR and FPR from Table 9. The figures show how increasing the *prior* improves the *BDR* significantly. As our confidence about the *prior* raises, the corresponding *BDR* increases to practical values.

Method	Dataset				
	Https	Tor	App-Fin	App-Comm	App-Social
Wknn	7.4	46.4	6.7	27.14	22.5
BindWknn	**15.3**	**49.2**	13.1	**54.4**	47.1
BindRF	14.6	37.4	**38.7**	25.3	**68.6**

Table 12: Base detection rate percentages in the open-world setting.

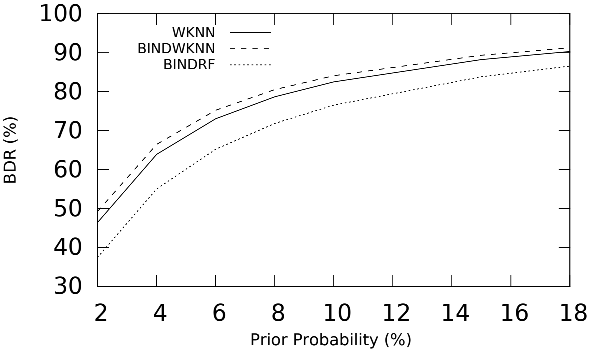

Figure 7: Increasing prior effect on BDR using the Tor dataset for open-world without defense.

4.3.6 *Adaptive fingerprinting*

We now present the experimental results of adaptive learning (AdaBind) discussed in §3.2.2. The experiment in Figure 9 shows the effect of concept drift on the model, and the BindDup dynamic update (re-training) process in Wfin. Here, the x-axis represents time (in days) and the y-axis represents accuracy (%). We consider 20 websites from the Https dataset with a training window of 16 traces per website for training the AdaBind model ($R = 16$, starting at day 1 to day 16). Then, a sliding window of 4 traces (starting at day 17) per website is considered for validating this model by testing its accuracy.

It is important to note the training and testing data are collected at different times, under different experimental settings. As the 4-day validating window slides, if the accuracy drops below a certain threshold (85% in this experiment), the model becomes obsolete. So, we re-train the model at that point (i.e., at day 33, 94, 119, and 148 as shown in the figure). This dynamic re-training mechanism improves the accuracy, resulting in values above the assigned threshold. The average accuracy of this approach is 92.6%.

Figure 9 also shows how the accuracy drops to low values if no update is considered. In this experiment, we train the model once in the beginning and use the 4-day sliding window to validate test traces. The resulting average accuracy of this static learning method is 76%, which illustrates the need for re-training the model to adapt for possible data drifts over time.

In addition, Figure 9 shows the same experiment where we apply the BindFup fixed update approach by re-training the model every 24 days instead of the dynamic update in BindDup. We use the same 4-day validating window as before. The figure shows how the model becomes more accurate and stable. Yet, this results in an extra training overhead due to unnecessary updates. The average accuracy of this approach is 93.3%, which is marginally better from the average accuracy of BindDup (92.6%). The number of updates in this experiment for BindFup is 8, which is twice as many as the number of updates in the dynamic update approach (BindDup). As discussed in § 4.3.4, a classifier may have large execution time, resulting in significantly large re-training cost. This shows the trade-off between performance and cost of re-training the model.

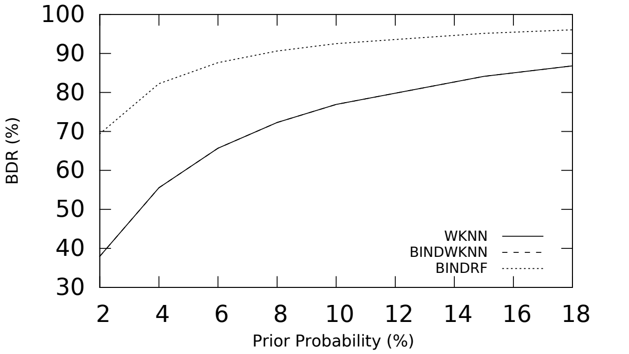

Figure 8: Increasing prior effect on BDR using the Tor dataset for open-world while applying the Tamaraw defense.

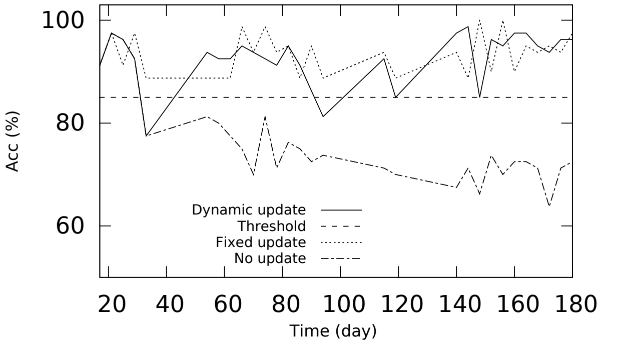

Figure 9: Adaptive Learning.

To see the effect of the training window (R), Figure 10 shows the BindDup dynamic update experiments when varying the value of R in the range $\{4, 8, 12, 16, 20\}$. If R is small, the number of training instances may not be enough to build a good model, and may lead to frequent updates. On the other hand, choosing large values of R incurs extra training overhead and may cause the model to miss some drifts in data. Table 13 shows the average accuracies and number of updates/re-trains for the experiments shown in Figure 10. When R increases, the average accuracy improves to a certain level, and then goes down. We obtained the best results when $R = 16$ with a moderate number of updates (i.e., 4 re-trains).

For the previous experiments which used SVM, we observed similar conclusions for the other datasets. We did not include them because of space limitations. In general, the adaptive learning algorithm can be applied to any classification approach.

5. DISCUSSION

In this paper, we introduced BIND, a new feature extraction and classification method for data analysis on encrypted network traffic with two case studies including Wfin and Afin. We discuss the challenges and limitations, resulting from the assumptions in our evaluation, as well as future work.

A study in Wfin [23] describes the effects of various assumptions on the evaluation results. Major assumptions include single-tabbed browsing or absence of other background noise, small time gap (or

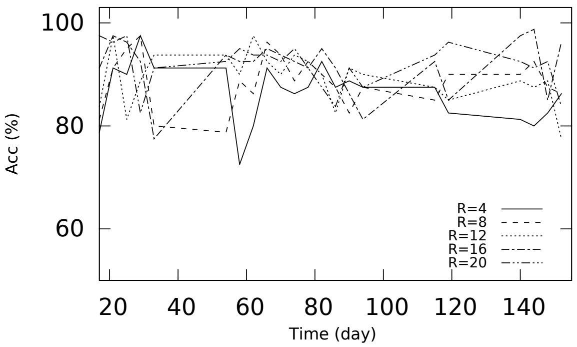

Figure 10: Dynamic update with different values of the training window (R)

R	4	8	12	16	20
Average accuracy (%)	86.6	89.3	89.9	**92.6**	91.7
Number of updates	10	7	5	4	2

Table 13: Average accuracies and number of updates with different values of the training window (R)

freshness) in data collection between training and test set, page load parsing, and replicability. Recent studies [18,38] tried to address these issues by evaluating classifiers in conditions with relaxed assumptions. In particular, a long time gap (or staleness) in data collection between training and testing sets can have a significant impact on classifier accuracy. This limitation is true for the BIND approach as well since similar base features that are affected with time, i.e., packet statistics such as length, sequence, and timing are used. The challenge can be addressed by periodically training a new model with fresh training data as introduced in this paper using AdaBind which models fingerprinting in an adaptive manner.

The AdaBind method updates the model with new training batches which requires a significant number of training instances. Furthermore, the re-training process assumes the availability of testing instance labels which may not be valid in certain cases. To address these challenges, in future we would like to identify the right point in the incoming stream from where we need to re-train the model incrementally (i.e., keeping old useful data) in an unsupervised manner (i.e., without labels). Hence, one of the future directions of BIND is to apply the concept of *Change Point Detection* (CPD) [19,20] to decide when to update the models in an unsupervised fashion and re-train incrementally.

The proposed methods in our paper assume sequential user access to end-nodes and ignore background noise, as mentioned in §2.1 regarding Wfin [23]. Nevertheless, these methods can be augmented with techniques relaxing such assumptions. We also note that such assumptions are applicable to Afin as well. In a smartphone, multiple apps may run background services, such as auto-sync, within the device that access the Internet periodically. Moreover, services offered by an app can change over time with newer versions released by developers periodically. Each updated version of an app may have dissimilar network signature or fingerprint, which could affect classifier performance as well. Furthermore, exploring different activities of an app would generate different network signatures compared to a signature obtained by merely launching it. One could use dynamic analysis techniques [4,32] to explore an app automatically for a better understanding of network behaviors. We leave these for future work.

6. CONCLUSION

We introduced, implemented, and evaluated BIND, a new data analysis method on encrypted network traffic for end-node identification. The method leverages dependence in packet sequences to extract characteristic features suitable for classification. In particular, we study two cases where our method is applicable: website fingerprinting and app fingerprinting. We empirically evaluate both these cases in the closed-world and open-world settings on various real-world datasets over HTTPS and Tor. Empirical results indicate the effectiveness of BIND in various scenarios including the realistic open-world setting. Our evaluations also include cases where defense mechanisms are applied on website and app fingerprinting. We showed how the proposed approach achieves a higher performance compared to other existing techniques. In addition, we introduced the AdaBind approach that addresses temporal changes in data patterns over time while performing traffic fingerprinting.

7. ACKNOWLEDGMENT

This material is based upon work supported by NSF under Award No. 1054629, AFOSR under Award No. FA9550-12-1-0077 and Award No. FA9550-14-1-0173, and NSA under Award No. H98230-15-1-0271.

References

[1] Alexa. The top visited sites on the web. http://www.alexa.com/.

[2] AlSabah, M., Bauer, K., and Goldberg, I. Enhancing tor's performance using real-time traffic classification. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 73–84.

[3] Ateniese, G., Hitaj, B., Mancini, L. V., Verde, N. V., and Villani, A. No place to hide that bytes won't reveal: Sniffing location-based encrypted traffic to track a user's position. In *Network and System Security*. Springer, 2015, pp. 46–59.

[4] Bhoraskar, R., Han, S., Jeon, J., Azim, T., Chen, S., Jung, J., Nath, S., Wang, R., and Wetherall, D. Brahmastra: Driving apps to test the security of third-party components. In *23rd USENIX Security Symposium (USENIX Security 14)* (2014), pp. 1021–1036.

[5] Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[6] Cai, X., Nithyanand, R., Wang, T., Johnson, R., and Goldberg, I. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), ACM, pp. 227–238.

[7] Cai, X., Zhang, X. C., Joshi, B., and Johnson, R. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 605–616.

[8] Chang, C.-C., and Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2* (2011), 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[9] Conti, M., Mancini, L. V., Spolaor, R., and Verde, N. V. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy* (2015), ACM, pp. 297–304.

[10] Conti, M., Mancini, L. V., Spolaor, R., and Verde, N. V. Analyzing android encrypted network traffic to identify user actions. *Information Forensics and Security, IEEE Transactions on 11*, 1 (2016), 114–125.

[11] CORTES, C., AND VAPNIK, V. Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297.

[12] DAI, S., TONGAONKAR, A., WANG, X., NUCCI, A., AND SONG, D. Networkprofiler: Towards automatic fingerprinting of android apps. In *INFOCOM, 2013 Proceedings IEEE* (2013), IEEE, pp. 809–817.

[13] DAVI, L., DMITRIENKO, A., SADEGHI, A.-R., AND WINANDY, M. Privilege escalation attacks on android. In *Information Security*. Springer, 2010, pp. 346–360.

[14] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. Tech. rep., DTIC Document, 2004.

[15] DOUGHERTY, J., KOHAVI, R., SAHAMI, M., ET AL. Supervised and unsupervised discretization of continuous features. In *Machine learning: proceedings of the twelfth international conference* (1995), vol. 12, pp. 194–202.

[16] DUDOROV, D., STUPPLES, D., AND NEWBY, M. Probability analysis of cyber attack paths against business and commercial enterprise systems. In *Proc. IEEE European Intelligence and Security Informatics Conf. (EISIC)* (2013), pp. 38–44.

[17] DYER, K. P., COULL, S. E., RISTENPART, T., AND SHRIMPTON, T. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Security and Privacy (SP), 2012 IEEE Symposium on* (2012), IEEE, pp. 332–346.

[18] GU, X., YANG, M., AND LUO, J. A novel website fingerprinting attack against multi-tab browsing behavior. In *Computer Supported Cooperative Work in Design (CSCWD), 2015 IEEE 19th International Conference on* (2015), IEEE, pp. 234–239.

[19] HAQUE, A., KHAN, L., AND BARON, M. SAND: Semi-supervised adaptive novel class detection and classification over data stream. In *Proc. 30th Conf. Artificial Intelligence (AAAI)* (2016), pp. 1652–1658.

[20] HAQUE, A., KHAN, L., BARON, M., THURAISINGHAM, B., AND AGGARWAL, C. Efficient handling of concept drift and concept evolution over stream data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)* (May 2016), pp. 481–492.

[21] HERRMANN, D., WENDOLSKY, R., AND FEDERRATH, H. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security* (2009), ACM, pp. 31–42.

[22] HITE, K. C., CICIORA, W. S., ALISON, T., BEAUREGARD, R. G., ET AL. System and method for delivering targeted advertisements to consumers, June 30 1998. US Patent 5,774,170.

[23] JUAREZ, M., AFROZ, S., ACAR, G., DIAZ, C., AND GREENSTADT, R. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), ACM, pp. 263–274.

[24] KIHL, M., ÖDLING, P., LAGERSTEDT, C., AND AURELIUS, A. Traffic analysis and characterization of internet user behavior. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on* (2010), IEEE, pp. 224–231.

[25] LIBERATORE, M., AND LEVINE, B. N. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security* (2006), ACM, pp. 255–263.

[26] MISKOVIC, S., LEE, G. M., LIAO, Y., AND BALDI, M. Appprint: automatic fingerprinting of mobile applications in network traffic. In *Passive and Active Measurement* (2015), Springer, pp. 57–69.

[27] PANCHENKO, A., LANZE, F., ZINNEN, A., HENZE, M., PENNEKAMP, J., WEHRLE, K., AND ENGEL, T. Website fingerprinting at internet scale. In *Proceedings of the 23rd Internet Society (ISOC) Network and Distributed System Security Symposium (NDSS 2016)* (2016). To appear.

[28] PANCHENKO, A., NIESSEN, L., ZINNEN, A., AND ENGEL, T. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society* (2011), ACM, pp. 103–114.

[29] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., ET AL. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research 12* (2011), 2825–2830.

[30] PLONKA, D. Flowscan: A network traffic flow reporting and visualization tool. In *LISA* (2000), pp. 305–317.

[31] RAYMOND, J.-F. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Designing Privacy Enhancing Technologies* (2001), Springer, pp. 10–29.

[32] SOUNTHIRARAJ, D., SAHS, J., GREENWOOD, G., LIN, Z., AND KHAN, L. Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps. In *Proceedings of the 19th Network and Distributed System Security Symposium* (2014).

[33] STÖBER, T., FRANK, M., SCHMITT, J., AND MARTINOVIC, I. Who do you sync you are?: smartphone fingerprinting via application behaviour. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks* (2013), ACM, pp. 7–12.

[34] TAYLOR, V., SPOLAOR, R., CONTI, M., AND MARTINOVIC, I. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *1st IEEE European Symposium on Security and Privacy (Euro S&P 2016)* (Mar 2016). To appear.

[35] VIDAS, T., VOTIPKA, D., AND CHRISTIN, N. All your droid are belong to us: A survey of current android attacks. In *WOOT* (2011), pp. 81–90.

[36] WANG, T., CAI, X., NITHYANAND, R., JOHNSON, R., AND GOLDBERG, I. Effective attacks and provable defenses for website fingerprinting. In *Proc. 23th USENIX Security Symposium (USENIX)* (2014).

[37] WANG, T., AND GOLDBERG, I. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society* (2013), ACM, pp. 201–212.

[38] WANG, T., AND GOLDBERG, I. On realistically attacking tor with website fingerprinting. Tech. rep., Technical Report 2015-08, CACR., 2015.

[39] WEI, T., ZHANG, Y., XUE, H., ZHENG, M., REN, C., AND SONG, D. Sidewinder targeted attack against android in the golden age of ad libraries. *Black Hat USA 2014* (2014).

[40] WRIGHT, C. V., COULL, S. E., AND MONROSE, F. Traffic morphing: An efficient defense against statistical traffic analysis. In *In Proceedings of the 16th Network and Distributed Security Symposium* (2009), IEEE, pp. 237–250.